AFRL-SN-WP-TR-2004-1019

AUTOMATIC DESIGN AND SYNTHESIS OF AUTOMATIC TARGET RECOGNITION (ATR) SYSTEMS USING LEARNING PARADIGMS

Bir Bhanu, Yingqiang Lin, and Krzysztof Krawiec

University of California
Bourns College of Engineering
Center for Research in Intelligent Systems
Riverside, CA 92521-0425

OCTOBER 2003

Final Report for 23 October 1999 – 22 October 2003

STINFO FINAL REPORT

SENSORS DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320

Contract Number: F33615-99-C-1440
Contractor: University of California, Riverside

/s/

KENNETH A. GRIER
Project Engineer
ATR & Fusion Algorithms Branch
Sensor ATR Technology Division

/s/

DALE E. NELSON, Ph.D.
Chief, ATR & Fusion Algorithms Branch
Sensor ATR Technology Division
Sensors Directorate

/s/

DAVID W. CHANDLER
Lieutenant Colonel, USAF
Deputy, Sensor ATR Technology Division
Sensors Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YY)* October 2003	**2. REPORT TYPE** Final	**3. DATES COVERED *(From - To)*** 10/23/1999 – 10/22/2003

4. TITLE AND SUBTITLE AUTOMATIC DESIGN AND SYNTHESIS OF AUTOMATIC TARGET RECOGNITION (ATR) SYSTEMS USING LEARNING PARADIGMS	**5a. CONTRACT NUMBER** F33615-99-C-1440
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62204F
6. AUTHOR(S) Bir Bhanu, Yingqiang Lin, and Krzysztof Krawiec	**5d. PROJECT NUMBER** 6095
	5e. TASK NUMBER 04
	5f. WORK UNIT NUMBER 04
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of California, Bourns College of Engineering, Center for Research in Intelligent Systems, Riverside, CA 92521-0425	**8. PERFORMING ORGANIZATION REPORT NUMBER** CRIS-1003
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Sensors Directorate, Air Force Research Laboratory, Air Force Materiel Command, Wright-Patterson AFB, OH 45433-7320	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** AFRL/SNAT
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-SN-WP-TR-2004-1019

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Report contains color.

14. ABSTRACT
This report investigates evolutionary computational techniques such as genetic programming (GP), coevolutionary genetic programming (CGP), linear genetic programming (LGP) and genetic algorithms (GA) to automate the synthesis and analysis of object detection and recognition systems. It shows the efficacy of evolutionary computation in synthesizing effective composite operators and composite features from domain-independent primitive image processing operations and primitive features for object detection and recognition. Smart crossover, smart mutation and a new fitness function based on minimum description length (MDL) principle are designed to improve the efficiency of genetic programming. A new MDL-based fitness function is proposed to improve the genetic algorithm's performance on feature selection for object detection and recognition. Results are shown using MSTAR SAR imagery.

15. SUBJECT TERMS
target detection/recognition, synthesis of recognition systems, feature selection, evolutionary computation, genetic programming, genetic algorithms, linear genetic programming

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Kenneth A. Grier
Unclassified	Unclassified	Unclassified	SAR	152	**19b. TELEPHONE NUMBER** *(Include Area Code)* (937) 904-9033

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Table of Contents

Acknowledgements **viii**

Chapter

1 Introduction **1**

2 Learning Composite Features for Object Detection Using Genetic Programming **6**

2.1 Motivation 7

2.2 Related Research 7

2.3 Technical Approach 8

2.3.1 Design Considerations 8

2.3.2 Selection, Crossover and Mutation 10

2.3.3 Steady-state and Generational Genetic Programming 12

2.4 Experiments 14

2.4.1 SAR Images 14

2.4.2 IR and RGB Images 24

2.4.3 Comparison with Example Region Selection 28

2.4.4 Comparison with a Simple ROI Extraction Algorithm 32

2.5 Summary 34

3 Improving Efficiency of Genetic Programming For Object Detection **35**

3.1 Motivation 35

3.2 Related Research 37

3.3 MDL-based Fitness Function 38

3.4 Technical Approach 40

3.5 Experiments 44

3.5.1 Road Extraction 44

3.5.2 Lake Extraction 49

3.5.3 River Extraction 50

3.5.4 Field Extraction 52

3.5.5 Tank Extraction 53

3.5.6 Comparison between Normal GP and Smart GP 54

3.6 Summary 56

4 GA-based Feature Selection for Object Detection 58

4.1 Motivation 58

4.2 Related Research 59

4.3 Technical Approach 61

4.3.1 Feature Evaluation 61

4.3.2 Various Criteria for Fitness Function 63

4.3.2.1 Fitness Function Based on MDL 63

4.3.2.2 Other Fitness Functions 64

4.3.3 System Description 65

4.3.3.1 CFAR Detector 65

4.3.3.2 Feature Extractor 66

4.3.3.3 GAs for Feature Selection 71

4.4 Experiments 72

4.4.1 MDL-based Fitness Function 73

4.4.2 Other Fitness Function 77

4.4.3 Comparison and Analysis 82

4.5 Summary 84

5 Learning Composite Features for Object Recognition Using Coevolutionary Genetic Programming 85

5.1 Motivation 86

5.2 Related Research 86

5.3 Technical Approach 88

5.3.1 Design Considerations 88

5.3.2 Selection, Crossover and Mutation 90

5.3.3 Generational Coevolutionary Genetic Programming 91

5.4 Experiments 92

5.4.1 Distinguish Object from Clutter 93

5.4.2 Recognize Objects 96

5.4.3 Discussions 102

5.5 Summary 102

6 Feature Synthesis for Recognition Using Linear Genetic Programming 104

6.1 Introduction 104

6.2 Motivation, Related Work and Contribution 105

6.2.1 Motivation 105

6.2.2 Related Work 105

6.2.3 Contributions 106

6.3 Technical Approach 107

6.3.1 Evolving Recognition Procedures 107

6.3.2 Representation of Feature Extraction Procedures 107

6.3.3 Cooperative Coevolution 109

6.3.4 Combining Cooperative Coevolution and Linear Genetic Programming 111

6.4 Experiments 112

6.4.1 Parameter Setting 112

6.4.2 Data and the Learning Task 113

6.5 Results 114

6.6 Summary 116

7 Coevolution and Linear Genetic Programming for Recognition-Further Extensions 119

7.1 Introduction 119

7.2 Related Work and Contributions 120

7.2.1 Coevolutionary Construction of Feature Extraction Procedures 121

7.2.2 Representation of Feature Extraction Procedures 123

7.2.3 Architecture of the Recognition System 125

7.3 Experimental Results 125

7.3.1 Scalability 127

7.3.2 Object Variants 128

7.4 Summary 130

8 **Conclusions** **131**

Bibliography **134**

Acknowledgements

Authors would like to thank Ken Grier, Dale Nelson and Lou Tamburino for their guidance and support during the course of this program. Many discussions held with Ed Zelnio, Tim Ross and Vince Velten were very helpful. This work was supported by contract F-33615-99-C-1440 from the Air Force Research Laboratory.

Chapter 1

Introduction

Object detection and recognition is one of the most important research areas in computer vision and pattern recognition. It has many applications in target recognition, video surveillance, etc.

The major task of object detection is to locate and extract regions that may contain objects in an image. It is an important intermediate step to object recognition. The extracted regions are called regions-of-interest (ROIs) or object chips. ROI extraction is very important to object recognition, since the size of an image is usually large, leading to the heavy computational burden of processing the whole image. By extracting ROIs, the computational cost of object recognition is greatly reduced, thus improving the recognition efficiency. This advantage is particularly useful to real-time applications, where the recognition speed is of prime importance. Also, by extracting ROIs, the recognition system can focus on the extracted regions that may contain potential objects and this can be very helpful in improving the recognition accuracy. Usually, in order to increase the probability of object detection, some false alarm ROIs, which don't contain an object, but a natural clutter or a man-made clutter, are allowed to pass object detection phase.

The task of object recognition is to reject the false alarm ROIs and recognize the kinds of objects contained in the ROIs. It is actually a signal-to-symbol problem of labeling perceived signals with one or more symbols [1]. A solution to this problem takes images or the features extracted from ROI images as input and outputs one or more symbols which are the labels of the objects in the images. Sometimes, the symbols may further represent the pose of the objects or the relations between different objects. These symbols are intended to capture some useful aspects of the input and in turn, permit some high level reasoning on the perceived signals.

As well known, object detection and recognition is really not an easy task. The quality of detection and recognition is heavily dependent on the kind and quality of features extracted from an image. The features used to represent an object are the key to the object detection and recognition. If useful features with good quality are unavailable to build an efficient representation of an object, the good detection and recognition results cannot be achieved no matter what detection and recognition algorithms are used. However, in most real images, there are always a variety of noises, which make the extraction of features difficult. More importantly, there are many kinds of features that can be extracted, what are the appropriate features for the current detection and recognition task or how to synthesize composite features particularly useful to the detection and recognition from the primitive features extracted from an image? There is no easy answer to these questions and the solutions are largely dependent on the intuitive instinct, knowledge, previous experience and even the bias of human image experts.

In recent years, with the advent of newer, much improved and inexpensive imaging technologies and the rapid expanding of the Internet, more and more images are becoming available. Recent developments in image collection platforms produce far more imagery than the declining ranks of image analysts are capable of handling due to the speed limit of human being. Relying entirely on human image experts to perform image analysis, processing and classification becomes more and more unrealistic. Building object detection and recognition systems to take advantage of the speed of computer is a viable and important solution to the increasing need of processing a large quantity of images efficiently.

Currently most of the object detection and recognition systems are manually developed and maintained by human experts. This traditional approach requires a human expert to select or synthesize a set of features to be used in detection and recognition. However, handcrafting a set of features requires human ingenuity and insight into the objects to be detected and recognized since it is very difficult to identify a set of features that characterize a complex set of objects to be encountered in the real-world application. Typically, many features are explored before object detection and recognition systems can be built. Many of these features may be correlated. To select a set of features which, when acting corporately, can give good performance is very time consuming and expensive. Sometimes, simple features (also called primitive features) directly extracted from images may not be effective in detecting and recognizing objects. At this point, synthesizing composite features useful for the current detection and recognition task from those simple ones becomes imperative. Traditionally, it is the human experts who synthesize features to be used. However, based on their knowledge, previous experience and limited by their bias and speed, human experts only consider a small number of conventional features and many unconventional features are totally ignored. Sometimes it is those unconventional features that yield very good detection and recognition performance.

Furthermore, after the features are selected or designed by human experts and incorporated into a detection/recognition system, they are fixed. The features used by the system are pre-determined and the system cannot generate new features useful to the current detection and recognition task on the fly based on the already available features, leading to the inflexibility of the system. Features useful to the detection and recognition of one kind of object or in the processing of one kind of imagery may not be effective in the detection and recognition of another kind of object or in the processing of another kind of imagery. Thus, the detection and recognition system needs thorough overhaul when applied to other types of images different from the one when the system was devised. This is very uneconomical.

Synthesizing effective new features from primitive features is equivalent to finding good points in the feature combination space where each point represents a combination of primitive features. Similarly, selecting an effective subset of features is equivalent to finding good points in the feature subset space where each point represents a subset of features. The feature combination space and feature subset space are huge and complicated and it is very difficult to find good points in such vast spaces unless one has an efficient search algorithm.

Hill climbing, gradient decent and simulated annealing (also called stochastic hill climbing) are widely used search algorithms. Hill climbing and gradient decent are efficient in exploring a unimodal space, but they are not suitable for finding global optimal points in a multi-modal space due to their high probability of being trapped in local optima. Thus, if the search space is a complicated and multi-modal space, they are unlikely to yield good search results. Simulated annealing has the ability to jump out of local optimal points, but it is heavily dependent on the starting point. If the starting point is not appropriately placed, it takes a long time, or even impossible, for simulated annealing to reach good points. Further more, in order to apply simulated annealing algorithm, the neighborhood of a point must be defined and the neighboring points should be somewhat similar. This requires some knowledge about the search space and it also requires the smoothness of the search space.

It is very difficult, if not impossible, to define the neighborhood of a point in the huge and complicated feature combination and feature subset spaces, since similar feature combination and similar feature subset may have very different object detection and recognition performances. Due to the lack of knowledge about these search spaces, genetic programming (GP) and genetic algorithm (GA) are employed in this report, since in order to apply GP, GA and LGP (Linear Genetic Programming), all that one needs to know are how to define individuals, how to define crossover and mutation operations on the individuals and how to evaluate individuals. GP, GA and LGP are very much capable of exploring huge complicated multi-modal spaces with unknown structures [18]. Maintaining a large population of individuals as multiple searching points, GP, GA and LGP explore the search spaces along different directions concurrently. With multiple searching points and the crossover and mutation operations' ability to immediately move a searching point from one portion of the search space to another far away portion, GP and GA are less likely to be trapped at local optimal points. All these characteristics greatly enhance the probability of finding global optimal points, although they cannot guarantee the finding of global optima. GP and GA are not random search algorithms. They are guided by the fitness of the individuals in the population. As search proceeds, the population is gradually adapted to the portion of the search space containing good points.

In this report, the techniques necessary to the automatic design of object detection and recognition systems are investigated. Here, the object detection and recognition system itself is the theme and the efficacy of evolutionary learning algorithms such as genetic programming, genetic algorithms and linear genetic programming in the feature generation and selection is studied. The advantage of incorporating learning is to avoid the time consuming process of feature selection and generation and automatically explore many unconventional features. Some unconventional features yield exceptionally good results in some cases, overcoming human expert's limitation of concentrating only on a small number of conventional features. The resulting systems are able to automatically generate features on the fly and cleverly select a good subset of features according to the types of object and images. The goal is to lower the cost of designing object detection and recognition systems and build more robust and flexible systems with human-competitive performance.

Chapter 2 discusses synthesizing composite features for object detection. Genetic programming (GP) is applied to the learning of composite features based on primitive features and primitive image processing operations. The primitive features and primitive image processing operations are domain-independent, not specific to any kind of imagery so that the proposed feature synthesis approach to be applied to a wide variety of images.

Chapter 3 concentrates on improving the efficiency of genetic programming. A fitness function based on the minimum description length (MDL) principle is proposed to address the well-known code bloat problem of GP while at the same time avoiding severe restriction on the GP search. The MDL fitness function incorporates the size of a composite operator into the fitness evaluation process to prevent it from growing too large, reducing possibility of overfitting during training and the computational expenses during testing. The smart crossover and smart mutation are proposed to identify the effective components of a composite operator and keep them from being disrupted by subsequent crossover and mutation operations to further improve the efficiency of GP.

In Chapter 4, genetic algorithm (GA) is applied to feature selection for distinguishing objects from natural clutter. Usually, GA is driven by a fitness function based on the performance of selected features. To achieve excellent performance during training, GA may select a large number of features. However, a large number features with excellent performance on training data may not perform well on unseen testing data due to the overfitting. Also, selecting more features means heavier computational burden during testing. In order to overcome this problem, an MDL-based fitness function is designed to drive GA. With MDL-based function incorporating the number of features selected into the fitness evaluation process, a small set of features is selected to achieve satisfactory performance during both training and testing.

Chapter 5 presents a method of learning composite feature vectors for object recognition. Coevolutionary genetic programming (CGP) is used to synthesize composite feature vectors based on the primitive features directly extracted from images. The experimental results using real SAR images show that CGP can evolve composite features that are more effective than the primitive features upon which they are built.

Chapter 6 presents linear genetic programming (LGP) in a cooperative coevolution framework for feature synthesis for object recognition. It provides a comparison between LGP and the standard genetic algorithm.

Chapter 7 provides further details and extends the ideas of chapter 6 and shows the results on eight classes of MSTAR SAR data.

Finally, Chapter 8 provides conclusions of this work.

Chapter 2

Learning Composite Features for Object Detection Using Genetic Programming

Designing automatic object detection systems is one of the important research areas in computer vision and pattern recognition [2, 3]. The major task of object detection is to locate and extract regions of an image that may contain potential objects so that the other parts of the image can be ignored. It is an intermediate step to object recognition and the regions extracted during detection are called regions-of-interest (ROIs). However, the quality of object detection is dependent on the type and quality of features extracted from an image. There are many features that can be extracted. The question is what are the appropriate features or how to synthesize features, particularly useful for detection, from the primitive features extracted from images. The answers to these questions are largely dependent on the intuitive instinct, knowledge, previous experience and even the bias of algorithm designers and experts in object detection by computer.

In this chapter, the effectiveness of genetic programming (GP) in synthesizing composite features, which are the output of composite operators, for object detection is investigated. A composite operator consists of primitive operators and primitive feature images. It can be viewed as a way of combining primitive operations on images. The basic approach is to apply a composite operator on the original image or primitive feature images generated from the original one; then the output image of the composite operator, called composite feature image, is segmented to obtain a binary image or mask; finally, the binary mask is used to extract the region containing the object from the original image. The individuals in our GP based learning are composite operators represented by binary tree whose internal nodes are the pre-specified primitive operators and the leaf nodes are the original image or the primitive feature images. The primitive feature images are pre-defined, and they are not the output of the pre-specified primitive operators.

2.1 Motivation

In most imaging applications, human experts design an approach to detect potential objects in images. The approach can often be dissected into some primitive operations on the original image or a set of related feature images obtained from the original one. It is the expert who, relying on his/her rich experience, figures out a smart way to combine these primitive operations to achieve good detection results. The task of synthesizing a good approach is equivalent to finding a good point in the space of *composite operators* formed by the combination of primitive operations.

Unfortunately, the ways of combining primitive operations are almost infinite. Limited by their knowledge, bias and speed, the human experts can only try a very limited number of conventional combinations, exploring just a very small portion of the composite operator space. Many unconventional combinations are regarded as nonsense, leaving a large portion of the composite operator space untouched. GP, however, may try many unconventional ways of combining primitive operations that may never be imagined by a human expert. Although these unconventional combinations are very difficult, if not impossible, to be explained by domain experts, in some cases, it is these unconventional combinations that yield exceptionally good detection results. In addition, the inherent parallelism of GP and the high speed of current computers allow the portion of the search space explored by GP to be much larger than that by human experts, enhancing the probability of finding an effective composite operator. The search performed by GP is not a random search. It is guided by the fitness of composite operators in the population. As the search proceeds, GP gradually shifts the population to the portion of the space containing good composite operators.

2.2 Related Research

Genetic programming, an extension of genetic algorithm, was first proposed by Koza [4] and has been used in image processing, object detection and object recognition. Harris et al. [5] applied GP to the production of high performance edge detectors for 1D signals and image profiles. The method is also extended to the development of practical edge detectors for use in image processing and machine vision. Poli [6] used GP to develop effective image filters to enhance and detect features of interest or to build pixel-classification-based segmentation algorithms. Bhanu and Lin [7] used GP to learn composite operators for object detection. Their initial experimental results showed that GP is a viable way of synthesizing composite operators from primitive operations for object detection. Stanhope and Daida [8] used GP to generate rules for target/clutter classification and rules for the identification of objects. To perform these tasks, previously defined feature sets are generated on various images and GP is used to select relevant features for analyzing these features. Howard et al. [9] applied GP to automatic detection of ships in low-resolution SAR imagery by evolving detectors. Roberts and Howard [10] used GP to develop automatic object detectors in infrared images.

Unlike the work of Stanhope and Daida [8], Howard et al. [9] and Roberts and Howard [10], the input and output of each node of the tree in our system are images, not real numbers. The primitive features defined in this chapter are more general and easier to compute than those used in [8, 9]. Unlike my previous work [7], the training in this chapter is not performed on a whole image, but on the selected regions of an image to greatly reduce the training time. Of course, training regions must be carefully selected and represent the characteristics of the training image [11]. Also, two other types of mutation are added to further increase the diversity of the population. Finally, more primitive feature images are employed. The primitive operators and primitive features designed in this chapter are very basic and domain-independent, not specific to a kind of imagery. Thus, the system and methodology can be applied to a wide variety of images such as synthetic aperture radar (SAR), infrared (IR) and RGB color video images.

2.3 Technical Approach

In the GP based approach of this report, individuals are composite operators represented by binary trees. The search space of GP is the space of all possible composite operators. The space is very large. To illustrate this, consider only a special kind of binary tree, where each tree has exactly 29 internal nodes and one leaf node and each internal node has only one child. For 17 primitive operators and only one primitive feature image, the total number of such trees is 17^{29}. It is extremely difficult to find good composite operators from this vast space unless one has a smart search strategy.

2.3.1 Design Considerations

There are five major design considerations, which involve determining the set of terminals, the set of primitive operators, the fitness measure, the parameters for controlling the evolutionary run, and the criterion for terminating a run.

- **The set of terminals**: The set of terminals used in this chapter are sixteen primitive feature images generated from the original image: the first one is the original image; the others are mean, deviation, maximum, minimum and median images obtained by applying templates of sizes 3×3, 5×5 and 7×7, as shown in Table 2.1. These images are the input to composite operators. GP determines which operations are applied on them and how to combine the results. To get the mean image, a template is translated across the original image and the average pixel value of the pixels covered by the template replaces the pixel value of the pixel covered by the central cell of the template. To get the deviation image, the pixel value difference between the pixel in the original image and its corresponding pixel in the mean image is computed. To get maximum, minimum and median images, a template is translated across the original image and the maximum, minimum and median pixel values of the pixels covered by the template replace the pixel value of the pixel covered by the central cell of the template, respectively.

Table 2.1. Sixteen primitive feature images used as the set of terminal.

No.	Primitive feature image	description	No.	Primitive feature image	description
0	PFIM0	Original image	8	PFIM8	5×5 maximum image
1	PFIM1	3×3 mean image	9	PFIM9	7×7 maximum image
2	PFIM2	5×5 mean image	10	PFIM10	3×3 minimum image
3	PFIM3	7×7 mean image	11	PFIM11	5×5 minimum image
4	PFIM4	3×3 deviation image	12	PFIM12	7×7 minimum image
5	PFIM5	5×5 deviation image	13	PFIM13	3×3 median image
6	PFIM6	7×7 deviation image	14	PFIM14	5×5 median image
7	PFIM7	3×3 maximum image	15	PFIM15	7×7 median image

- **The set of primitive operators:** A primitive operator takes one or two input images, performs a primitive operation on them and stores the result in a resultant image. Currently, 17 primitive operators are used by GP to evolve composite operators, as shown in Table 2.2, where A and B are input images of the same size and c is a constant (ranging from –20 to 20) stored in a primitive operator. For operators such as ADD, SUB, MUL, etc. that take two images as input, the operations are performed on the pixel-by-pixel basis. In the operators MAX, MIN, MED, MEAN and STDV, 3×3, 5×5 or 7×7 neighborhood are used with equal probability.

- **The fitness measure:** The fitness value of a composite operator is computed in the following way. Suppose G and G' are foregrounds in the ground truth image and the resultant image of the composite operator respectively. Let $n(X)$ denote the number of pixels within region X, then $Fitness = n(G \cap G') / n(G \cup G')$. The fitness value is between 0 and 1. If G and G' are completely separated, the value is 0; if G and G' are completely overlapped, the value is 1.

- **Parameters and termination:** The key parameters are the population size M, the number of generation N, the crossover rate, the mutation rate and the fitness threshold. The GP stops whenever it finishes the pre-specified number of generations or whenever the best composite operator in the population has fitness value greater than the fitness threshold.

2.3.2 **Selection, Crossover and Mutation**

GP searches through the space of composite operator to generate new composite operators, which may be better than the previous ones. By searching through the composite operator space, GP adapts the population of composite operators from generation to generation and improves the overall fitness of the whole population. More importantly, GP may find an exceptionally good composite operator during the search. The search is done by performing selection, crossover and mutation operations. The initial population is randomly generated and the fitness of each individual is evaluated.

- **Selection:** The selection operation involves selecting composite operators from the current population. In this chapter, tournament selection is used, where a number of individuals are randomly selected from the current population and the one with the highest fitness value is copied into the new population. The size of tournament is 5.

- **Crossover:** To perform crossover, two composite operators are selected on the basis of their fitness values. The higher the fitness value, the more likely the composite operator is selected for crossover. These two composite operators are called parents. One internal node in each of these two parents is randomly selected, and the two subtrees rooted at these two nodes are exchanged between the parents to generate two new composite operators, called offspring. The offspring are composed of subtrees from their parents. If two composite operators are somewhat effective in detection, then some of their parts probably have some merit. The reason that an offspring may be better than the parents is that recombining randomly chosen parts of effective composite operators may yield a new composite operator that is more effective in detection.

It is easy to see that the size of one offspring (i.e., the number of nodes in the binary tree representing the offspring) may be greater than those of both parents. So if we do not control the size of a composite operator by implementing crossover in this simple way, the sizes of composite operators will become larger and larger as GP proceeds. This is the well-known code bloat problem of GP. It is a very serious problem, since when the size becomes too large; it will take a long time to execute a composite operator, greatly reducing the search speed of GP. Further, large-size composite operators may overfit training data by approximating various noisy components of an image. Although the results on the training image may be very good, the performance on unseen testing images may be bad. Also, large composite operators take up a lot of computer memory. Due to the finite computer resources and the desire to achieve a good running speed (efficiency) of GP, we must limit the size of a composite operator by specifying a maximum size. Currently, a simple method is used to address this problem by setting a size limit on the size of a composite operator. If the size of an offspring exceeds the maximum size allowed, the crossover operation is performed again until the sizes of both offspring are within the limit. Although this simple method guarantees that the size of a composite operator won't exceed the size limit, the hard size limit greatly restricts the search performed by GP, since after randomly selecting a crossover point in one composite operator, GP cannot select some nodes of the other composite operator as crossover points in order to guarantee that both offspring won't exceed the size limit. Restricting the search greatly reduces the efficiency of GP, making it less likely to find good composite operators. In chapter 3, a new fitness function based on minimum description length (MDL) principle will be used to incorporate the size of a composite operator into the fitness evaluation process to prevent the code bloat without imposing severe restriction on the GP search. The essential idea of applying MDL-based fitness function is to find a balance point between the above two conflicting factors.

Table 2.2. Seventeen primitive operators.

No.	Operator	Description
1	ADD (A, B)	Add images A and B.
2	SUB (A, B)	Subtract image B from A.
3	MUL (A, B)	Multiply images A and B.
4	DIV (A, B)	Divide image A by image B (If the pixel in B has value 0, the corresponding pixel in the resultant image takes the maximum pixel value in A).
5	MAX2 (A, B)	The pixel in the resultant image takes the larger pixel value of images A and B.
6	MIN2 (A, B)	The pixel in the resultant image takes the smaller pixel value of images A and B.
7	ADDC (A)	Increase each pixel value by c.
8	SUBC (A)	Decrease each pixel value by c.
9	MULC (A)	Multiply each pixel value by c.
10	DIVC (A)	Divide each pixel value by c.
11	SQRT (A)	For each pixel with value v, if v ≥ 0, change its value to $\sqrt{v}$. Otherwise, to $-\sqrt{-v}$.
12	LOG (A)	For each pixel with value v, if v ≥ 0, change its value to ln(v). Otherwise, to –ln(-v).
13	MAX (A)	Replace the pixel value by the maximum pixel value in a 3×3, 5×5 or 7×7 neighborhood.
14	MIN (A)	Replace the pixel value by the minimum pixel value in a 3×3, 5×5 or 7×7 neighborhood.
15	MED (A)	Replace the pixel value by the median pixel value in a 3×3, 5×5 or 7×7 neighborhood.
16	MEAN (A)	Replace the pixel value by the average pixel value of a 3×3, 5×5 or 7×7 neighborhood.
17	STDV (A)	Replace the pixel value by the standard deviation of pixels in a 3×3, 5×5 or 7×7 neighborhood.

- **Mutation:** In order to avoid premature convergence, mutation is introduced to randomly change the structure of some individuals to maintain the diversity of the population. Composite operators are randomly selected for mutation. There are three types of mutation invoked with equal probability:

    1. Randomly select a node of the binary tree representing a composite operator and replace the subtree rooted at this node, including the node selected, by another randomly generated binary tree
    2. Randomly select a node of the binary tree representing a composite operator and replace the primitive operator stored in the node with another primitive operator of the same number of inputs as the replaced one. The replacing primitive operator is selected at random from all the primitive operators with the same number of input as the replaced one.
    3. Randomly select two subtrees within a composite operator and swap them. Of course, neither of the two sub-trees can be a sub-tree of the other.

2.3.3 Steady-state and Generational Genetic Programming

Both steady-state and generational genetic programming are used in this chapter. In *steady-state GP*, two parent composite operators are selected on the basis of their fitness for crossover. The offspring of this crossover replace a pair of composite operators with the smallest fitness values. The two offspring are executed immediately and their fitness values are recorded. Then another two parent composite operators are selected for crossover. This process is repeated until crossover rate is satisfied. Finally, mutation is applied to the resulting population and the mutated composite operators are executed and evaluated. The above cycle is repeated from generation to generation. In *generational GP*, two composite operators are selected on the basis of their fitness values for crossover and generate two offspring. The two offspring are not put into the current population and won't participate in the following crossover operations on the current population. The above process is repeated until crossover rate is satisfied. Then, mutation is applied to the composite operators in the current population and the offspring from crossover. After mutation is done, selection is applied to the current population to select some composite operators. The number of composite operators selected must meet the condition that after combining with the composite operators from crossover, a new population of the same size as the old one is resulted. Finally, combine the composite operators from crossover with those selected from the old population to get a new population and the next generation begins. In addition, an elitism replacement method is adopted to keep the best composite operator from generation to generation.

- **Steady-state Genetic Programming:**

```
0. randomly generate population P of size M and evaluate each composite
   operator in P.
```

```
1.  for gen = 1 to N do   // N is the number of generation
2.     keep the best composite operator in P.
3.     repeat
4.          select 2 composite operators from P based on their fitness values for
            crossover.
5.          select 2 composite operators with the lowest fitness values in P for
            replacement.
6.          perform crossover operation and let the 2 offspring replace the 2
            composite operators selected for replacement.
7.          execute the 2 offspring and evaluate their fitness values.
8.     until crossover rate is met.
9.     perform mutation on each composite operator with probability of muta-
       tion rate and evaluate mutated composite operators.
10.    After crossover and mutation, a new population P' is generated.
11.    let the best composite operator from population P replace the worst
       composite operator in P' and let P = P'.
12.    if the fitness value of the best composite operator in P is above fitness
       threshold value then
13.       stop.
       endif
    endfor  // loop 1
```

- **Generational Genetic Programming:**

```
0.  randomly generate populations of size M and evaluate each composite
    operator in P.
1.  for gen = 1 to N do   // N is the number of generation
2.     keep the best composite operator in P.
3.     perform crossover on the composite operators in P until crossover rate
       is satisfied and keep all the offspring from crossover separately.
4.     perform mutation on the composite operators in P and the offspring
       from crossover with the probability of mutation rate.
5.     perform selection on P to select some composite operators. The number
       of selected composite operators must be M minus the number of com-
       posite operators from crossover.
6.     combine the composite operators from crossover with those selected
       from P to get a new population P' of the same size as P.
7.     evaluate offspring from crossover and the mutated composite opera-
       tors.
8.     let the best composite operator from P replace the worst composite op-
       erator in P' and let P = P'.
9.     if the fitness of the best composite operator in P is above fitness
       threshold then
10.       stop.
```

```
        *endif*
    *endfor  // loop 1*
```

2.4 Experiments

Various experiments are performed to test the efficacy of genetic programming in extracting regions of interest from real synthetic aperture radar (SAR) images, infrared (IR) images and RGB color images. The size of SAR images is 128×128, except the tank SAR images whose size is 80×80, and the size of IR and RGB color images is 160×120. GP (in subsections 1.3.1 and 1.3.2) is not applied to a whole training image, but only to a region or regions carefully selected from the training image, to generate composite operators. The best-generated composite operator is then applied to the whole training image and to some other testing images to evaluate it. The advantage of performing training on a small selected region is that it can greatly reduce the training time, making it practical for the GP system to be used as a subsystem of other learning systems, which improve the efficiency of GP by adapting the parameters of GP system based on its performance. The experiments show that if training regions are carefully selected from a training image, the best composite operator generated by GP is effective. In the following experiments in sections 1.3.1 and 1.3.2, the parameters are: population size (100), the number of generations (70), the fitness threshold value (1.0), the crossover rate (0.6), the mutation rate (0.05), the maximum size of composite operator (30), and the segmentation threshold (0). In each experiment, GP is invoked ten times with the same parameters and the same training region(s). The coordinate of the upper left corner of an image is (0, 0). The ground truth is used only during training; it is not needed during testing. It is used in testing only for evaluating the performance of the composite operator on testing images.

2.4.1 SAR Images

Five experiments are performed with real SAR images. The experimental results from one run and the average performance of ten runs are reported in Table 2.3. The results of the run in which GP finds the best composite operator among the composite operators found in all ten runs are reported. The first two rows show the fitness value of the best composite operator and the population fitness value (average fitness value of all the composite operators in the population) on training region(s) in the initial and final generations in the selected run. The numbers in the parenthesis in the "f_{op}" columns are the fitness values of the best composite operators on the whole training image (numbers with a * superscript) and other testing images in their entirety. The last two rows show the average values of the above fitness values over all ten runs. The regions extracted during the training and testing by the best composite operator from the selected run are shown in the following examples.

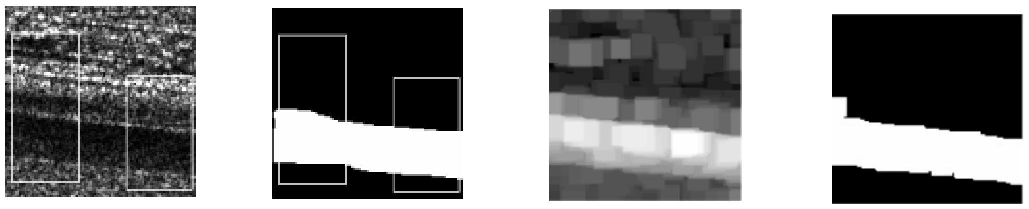
- **Example 1 — Road extraction:** Three images contain road, the first one contains horizontal paved road and field (Fig 2.1(a)); the second one contains unpaved road and field (Fig 2.8(a)); the third one contains vertical paved road and grass (Fig 2.8(d)). Training is done on the training regions of the training image shown in Figure 2.1(a) and testing is performed on the whole training image and testing images. There are two training regions, locating from (5, 19) to (50, 119) and from (82, 48) to (126, 124), respectively. Figure 2.1(b) shows the ground truth provided by the user and the training regions. The white region corresponds to the road and only the ground truth in the training regions is used in the evaluation during the training. Figure 2.2 shows the sixteen primitive feature images of the training image.

The generational GP is used to synthesize a composite operator to extract the road and the results of the 7th run are reported. The fitness value of the best composite operator in the initial population is 0.60 and the population fitness value is 0.27. The fitness value of the best composite operator in the final population is 0.94 and

Table 2.3. The performance on various examples of SAR images.
(f_{op} = fitness of the best composite operator, f_p = fitness of population, *: indicate finess on training images, $f_{initial}$ = fitness in the initial generation, f_{final} = fitness in the final population)

	Road		Lake		River		Field		Tank	
	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p
$f_{initial}$	0.60	0.27	0.62	0.30	0.59	0.19	0.52	0.38	0.65	0.17
f_{final}	0.94 (0.90*, 0.90, 0.93)	0.93	0.99 (0.95*, 0.97)	0.95	0.89 (0.72*, 0.83)	0.86	0.78 (0.88*, 0.81)	0.77	0.88 (0.88*, 0.84)	0.87
Ave. $f_{initial}$	0.47	0.26	0.64	0.32	0.49	0.18	0.53	0.38	0.49	0.16
Ave. f_{final}	0.82	0.81	0.93	0.92	0.82	0.77	0.73	0.72	0.85	0.83

the population fitness value is 0.93. Figure 2.1(c) shows the output image of the best composite operator on the whole training image and Figure 2.1(d) shows the binary image after segmentation. The output image has both positive pixels in brighter shade and negative pixels in darker shade. Positive pixels belong to the region to be extracted. The fitness value of the extracted ROI is 0.90. The best composite operator has 27 nodes and its depth is 16. It has five leaf nodes, three contain 5×5 median image and the other two contain 7×7 median image. The median images have less speckle noise, since median filtering is effective in eliminating speckle noises. It is shown in Figure 2.3, where PFIM14 and PFIM15



(a) paved road vs. field (b) ground truth (c) composite feature image (d) ROI extracted

Figure 2.1. Training SAR image containing road.

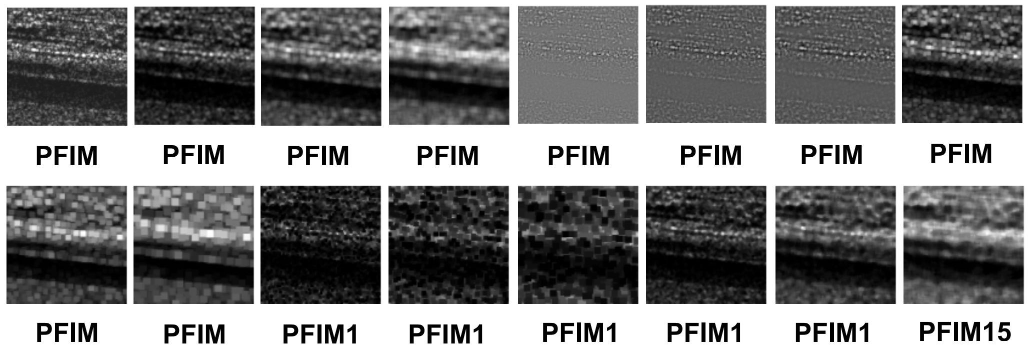


Figure 2.2. Sixteen primitive feature images of training SAR image containing road.

are 5×5 and 7×7 median images, respectively. Figure 2.4 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP explore the composite operator space. It is obvious that GP gradually shifts the population to the regions of space containing good composite operators.

```
(MAX (MAX (MIN (DIVC (DIV
(ADDC (ADD (ADDC (ADD (SUBC
(ADDC (ADD (SUBC (STDV (MAX
(SUBC PFIM15)))) (MAX (SUBC
PFIM14))))) (MAX (SUBC
PFIM14)))) (MAX (SUBC PFIM14))))
PFIM15)))))
```

Figure 2.3. Learned composite operator tree in LISP notation.

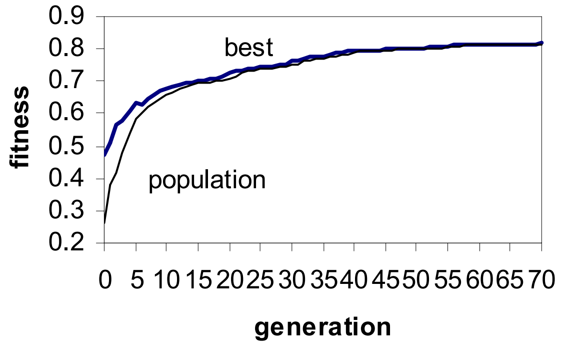


Figure 2.4. Fitness versus generation (road vs. field).

10 best composite operators are obtained in the initial and final generations of 10 runs, respectively. After computing the percentage of each primitive operator and primitive feature image among the total number of internal nodes (representing primitive operator) and total number of leaf nodes (representing primitive feature image) of 10 best composite operators, the utility of these primitive operators and primitive feature images in the initial and final populations is obtained, which is shown in Figure 2.5. Compared to those in the final population, the utilities of

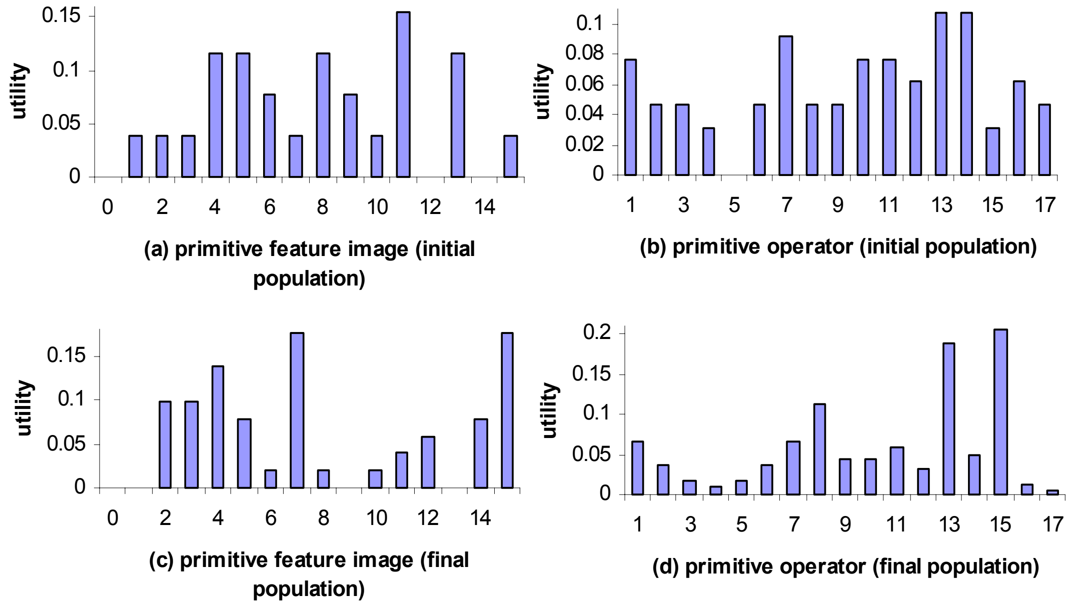


Figure 2.5. Utility of primitive operators and primitive feature images.

primitive feature images and primitive operators are relatively uniformly distributed in the initial population. In the final population, primitive feature images PFIM7 (3×3 maximum image) and PFIM15 (7×7 median image) and primitive operator MED (primitive operator 15) have the highest frequency of utility. As well known, median filter is effective in eliminating speckle noise in SAR images. Figure 2.6 shows the output image of each node of the best composite operator shown in Figure 2.3. The primitive operators in Figure 2.6 are connected by arrow. The operator at the tail of an arrow provides input to the operator at the head of the arrow. After segmenting the output image of a node, the ROI (shown as the white region) extracted by the corresponding subtree rooted at the node is obtained. The extracted ROIs and their fitness values are shown in Figure 2.7. If an output image has positive pixels only (for example, PFIM14 has positive pixels only), everything is extracted and the fitness is 0.25.

The composite operator obtained in the above training is applied to the other two real SAR images shown in Figure 2.8(a) and 2.8(d). Figure 2.8(b) and 2.8(e) show the output of the composite operator and Figure 2.8(c) shows the region extracted from Figure 2.8(a). The fitness value of the region is 0.90. Figure 2.8(f)

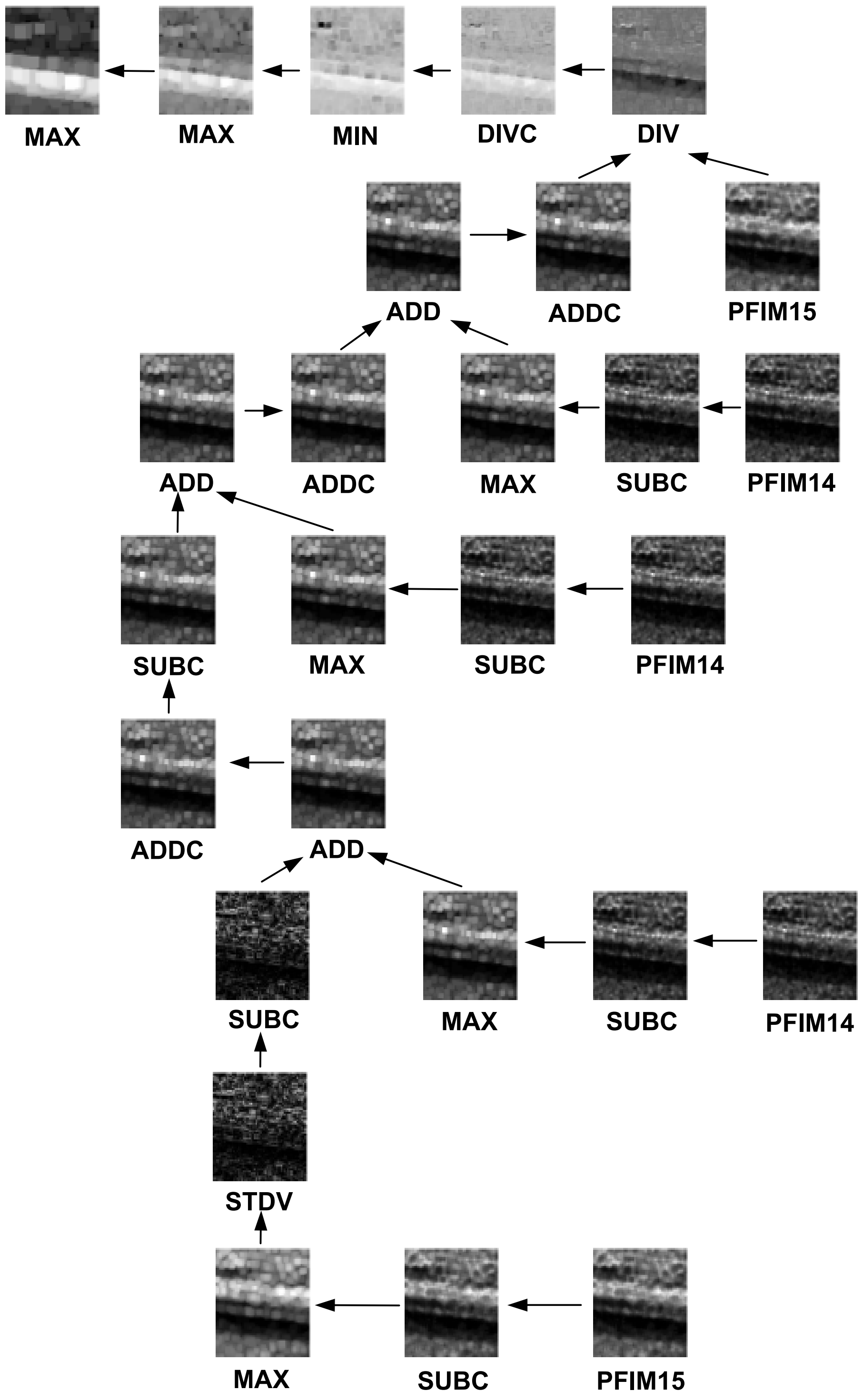


Figure 2.6. Feature images output by the nodes of the best composite operator.

shows the region extracted from Figure 2.8(d) and the fitness value of the region, which is 0.93.

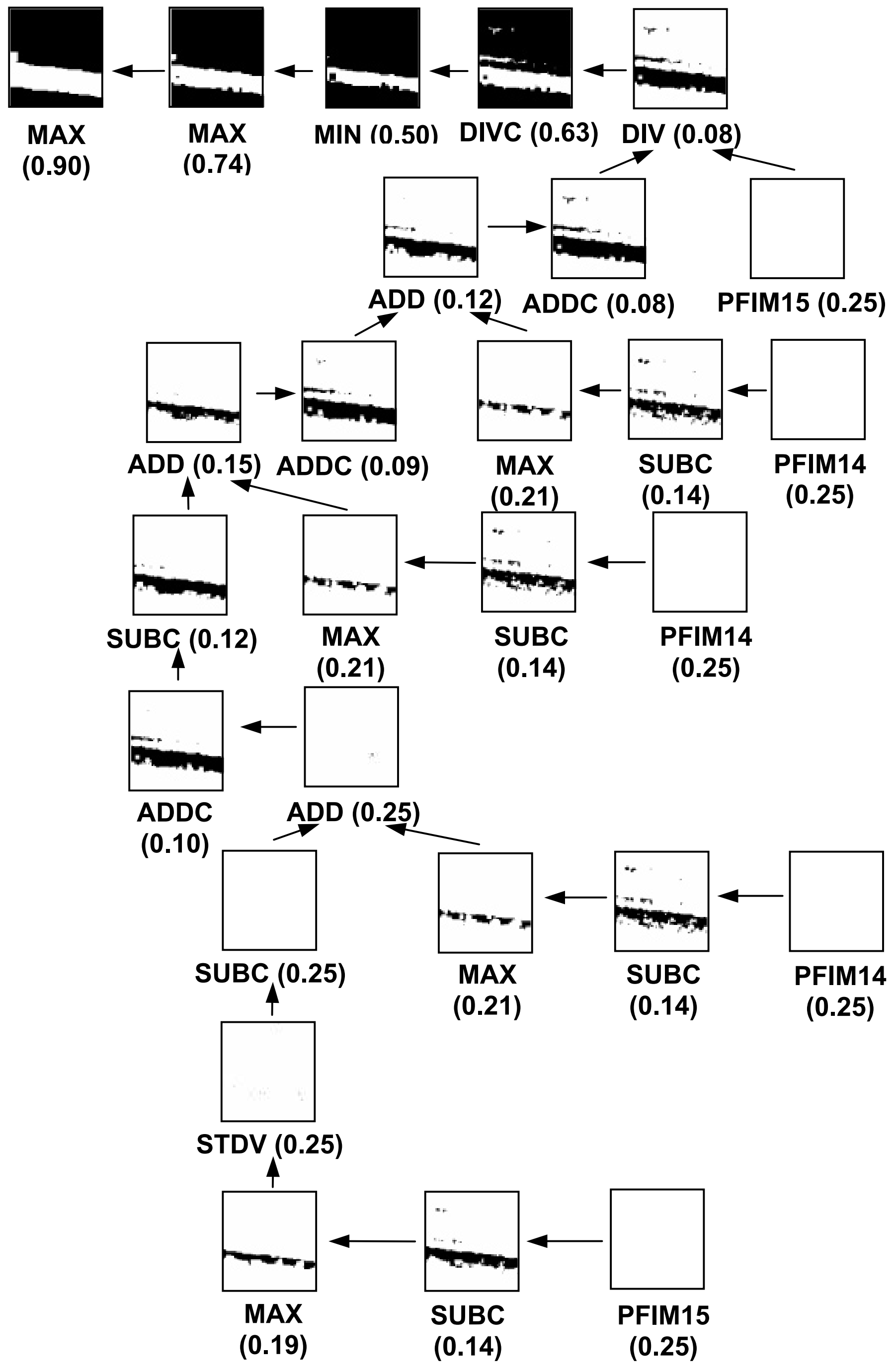


Figure 2.7. ROIs extracted from the output images of the nodes of the best composite operator. (The fitness value is shown for the entire image.)

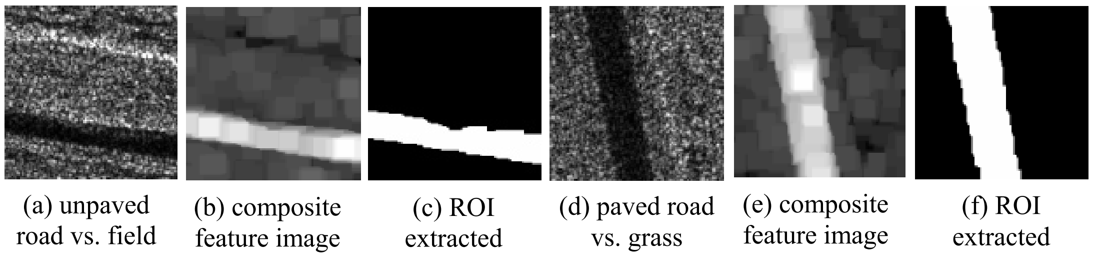

(a) unpaved road vs. field
(b) composite feature image
(c) ROI extracted
(d) paved road vs. grass
(e) composite feature image
(f) ROI extracted

Figure 2.8. Testing SAR images containing road.

- **Example 2 — Lake extraction:** Two SAR images contain lake (Fig 2.9(a), 2.10(a)), the first one contains a lake and field, and the second one contains a lake and grass. Figure 2.9(a) shows the original training image containing lake and field and the training region from (85, 85) to (127, 127). Figure 2.9(b) shows the ground truth provided by the user. The white region corresponds to the lake to be extracted. Figure 2.10(a) shows the image containing lake and grass.

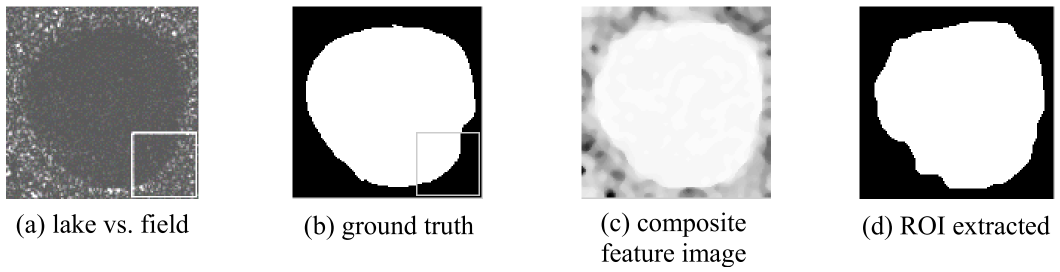
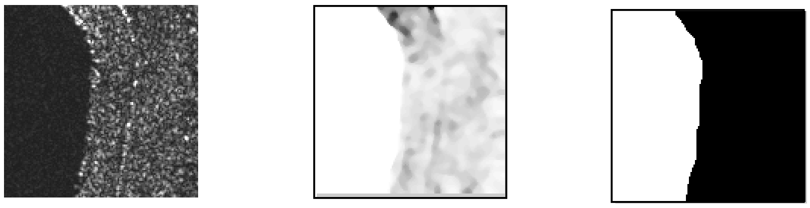


Figure 2.9. Training SAR image containing lake.

The steady-state GP is used to generate the composite operator and the results of the 4th run are reported. The fitness value of the best composite operator in the initial population is 0.62 and the population fitness value is 0.30. The fitness value of the best composite operator in the final population is 0.99 and the population fitness value is 0.95. Figure 2.9(c) shows the output image of the best composite operator on the whole training image and Figure 2.9(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.95. The composite operator is applied to the testing image containing lake and grass. Figure 2.10(b) shows the output of the composite operator and Figure 2.10(c) shows the region extracted. The fitness of the region is 0.97.

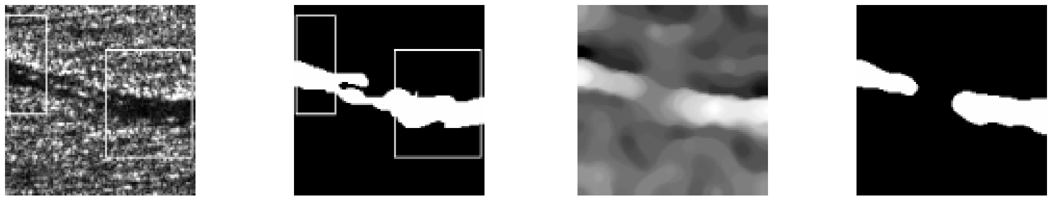
- **Example 3 — River extraction:** Two SAR images contain river and field. Figure 2.11(a) and 2.11(b) show the original training image and the ground truth provided by the user. The white region in Figure 2.11(b) corresponds to the river to be extracted. The training regions are from (68, 31) to (126, 103) and from (2, 8) to (28, 74). The testing SAR image is shown in Figure 2.14(a).

The steady-state GP is used to generate composite operators and the results of the first run are reported. The fitness value of the best composite operator in the initial population is 0.59 and the population fitness value is 0.19. The fitness value of the best composite operator in the final population is 0.89 and the population fitness value is 0.86. Figure 2.11(c) shows the output image of the best composite operator on the whole training image and Figure 2.11(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.72. The best composite operator has 30 nodes and its depth is 23. It has four leaf nodes, three contain 5×5 mean image and the other one contains 3×3 mean image.



(a) lake vs. grass (b) composite feature image (c) ROI extracted

Figure 2.10. Testing SAR image containing lake.



(a) river vs. field (b) ground truth (c) composite feature image (d) ROI extracted

Figure 2.11. Training SAR image containing river.

```
(MULC (MED (MED (MED
(MED (MED (MED (MED (MED
(MED (MED (MED (MED (MED
(MED (MED (MIN (ADDC (LOG
(ADD (MAX (MIN (MULC
PFIM2))) (DIV (MIN (MULC
(MED (MIN (MAX (SUB PFIM2
(MULC PFIM2)))))))
PFIM1)))))))))))))))))
```

Figure 2.12. Learned composite operator tree in LISP notation.

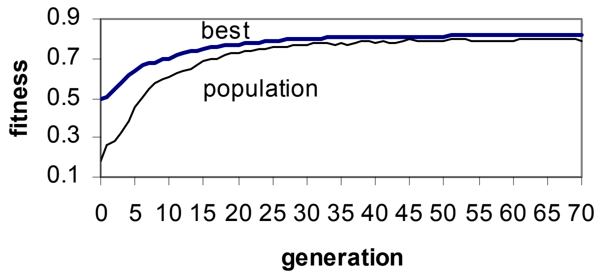


Figure 2.13. Fitness versus generation (river vs. field).

There are more than ten MED operators that are very useful in eliminating speckle noises. It is shown in Figure 2.12. Figure 2.13 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP explores the composite operator space.

The composite operator is applied to the testing image containing a river and field. Figure 2.14(b) shows the output of the composite operator and Figure 2.14(c) shows the region extracted from Figure 2.14(a) and the fitness value of the extracted region is 0.83. There are some islands in the river and these islands along with the river around them are not extracted, since the islands look similar to the field.

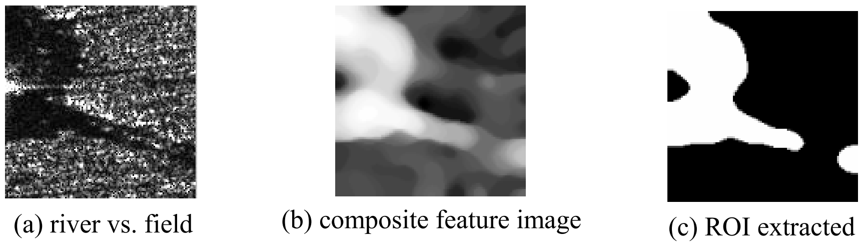


Figure 2.14. Testing SAR image containing river.

- **Example 4 — Field extraction:** Two SAR images contain field and grass. Figure 2.15(a) and 2.15(b) show the original training image and the ground-truth. The training regions are from (17, 3) to (75, 61) and from (79, 62) to (124, 122). Extracting field from a SAR image containing field and grass is considered as the most difficult task among the five experiments, since the grass and field are similar to each other and some small regions between grassy areas are actually field pixels.

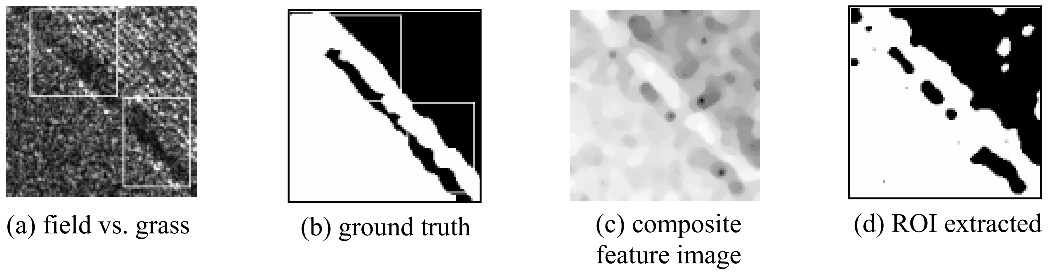


Figure 2.15. Training SAR image containing field.

The generational GP was used to generate composite operators and the results from the 7th run are reported. The fitness value of the best composite operator in the initial population is 0.52 and the population fitness value is 0.38. The fitness value of the best composite operator in the final population is 0.78 and the population fitness value is 0.77. Figure 2.15(c) shows the output image of the best composite operator on the whole training image and Figure 2.15(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.88. The composite operator is applied to the testing image containing field and grass shown in Figure 2.16(a). Figure 2.16(b) shows the output of the composite operator and Figure 2.16(c) shows the region extracted from Figure 2.16(a). The fitness value of the region is 0.81.

Example 5 — Tank extraction: In this subsection, GP is applied to synthesize features for the detection of military targets and this application is of special importance. The military targets used in this chapter are T72 tanks. Their SAR images are taken under different depression and azimuth angles and the size of the images is 80×80. The training image contains T72 tank under depression angle 17° and azimuth angle 135°, which is shown in Figure 2.17(a). The training region is from (19, 17) to (68, 66). The testing SAR image contains a T72 tank

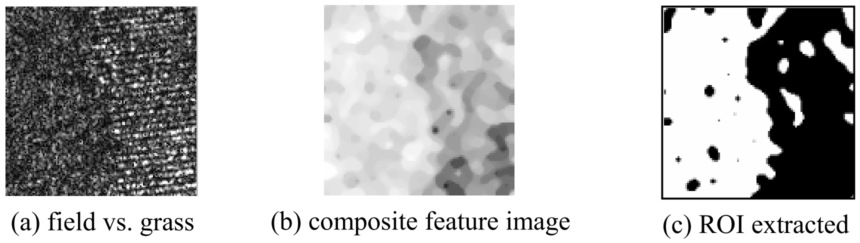


Figure 2.16. Testing SAR image containing field.

under depression angle 20° and azimuth angle 225°, which is shown in Figure 2.20(a). The ground-truth is shown in Figure 2.17(b).

The generational GP is applied to synthesize composite operators for tank detection and the results from the 6th run are reported. The fitness value of the best composite operator in the initial population is 0.65 and the population fitness value is 0.17. The fitness value of the best composite operator in the final population is 0.88 and the population fitness value is 0.87. Figure 2.17(c) shows

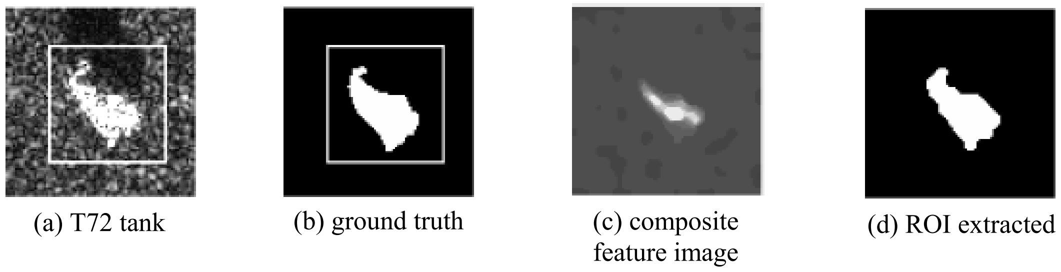


Figure 2.17. Training SAR image containing tank.

the output image of the best composite operator on the whole training image and Figure 2.17(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.88. The best composite operator has 28 nodes and its depth is 17. It has four leaf nodes, two contain the 3×3 minimum image, one contains 7×7 maximum image and the final one contains 7×7 minimum image. It is shown in Figure 2.18. Figure 2.19 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP proceeds.

```
(MED (MED (MUL (MIN PFIM10)
(MUL (MAX PFIM12) (MIN2 (MAX
(SUBC (DIVC (MIN (MEAN
PFIM9))))) (SUBC (MED (SUBC
(MAX (MAX (SUBC (MAX (MAX
(SUBC (MAX (MAX (SUBC
PFIM10)))))))))))))))))
```

Figure 2.18. Learned composite operator tree in LISP notation.

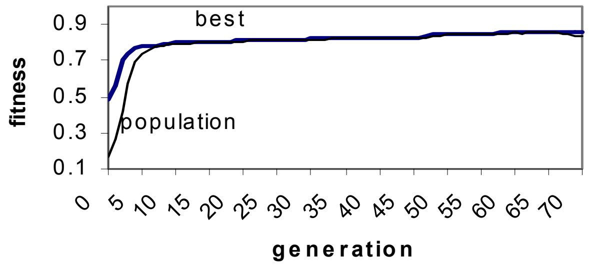
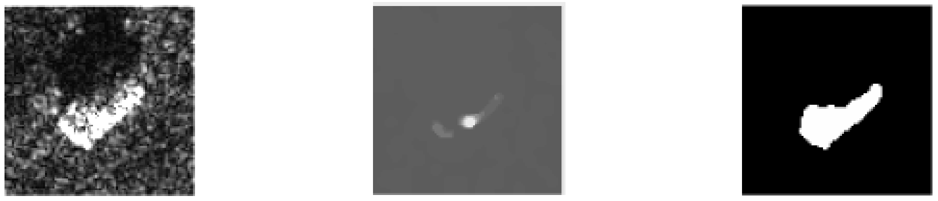


Figure 2.19. Fitness versus generation (T72 tank).

The composite operator is applied to the testing image containing T72 tank under depression angle 20° and azimuth angle 225°. Figure 2.20(b) shows the output of the composite operator and Figure 2.20(c) shows the region corresponding to the tank. The fitness of the extracted ROI is 0.84. The results show that GP is very much capable of synthesizing composite operators for military target detection. With more and more SAR images collected by satellites and airplanes, it is impractical for human experts to scan each SAR image to find military targets. Applying the synthesized composite operators on these images, regions containing potential targets can be quickly detected and passed on to an automatic target recognition system or human experts for further examination. Concentrating on the regions of interest, the human experts and recognition systems can perform recognition task more effectively and more efficiently.



(a) T72 tank (b) composite feature image (c) ROI extracted

Figure 2.20. Testing SAR image containing tank.

2.4.2 IR and RGB Color Images

One experiment is performed with IR images and one is performed with RGB color images. The experimental results from one run and the average performance of ten runs are reported in Table 2.4. As in Subsection 2.4.1, the results of the run in which GP finds the best composite operator among the composite operators found in all the ten runs are reported. The regions extracted during the training and testing by the best composite operator from the selected run are shown in the following examples.

People extraction in IR images: In IR images, pixel values correspond to the temperature in the scene. There are four IR images with one used in training and the other three used in testing. Figure 2.21(a) and (b) show the training image and the ground truth. There are two training regions from (59, 9) to (106, 88) and from (2, 3) to (21, 82), respectively. The left training region contains no pixel belonging to the person. The reason for selecting it during training is that there are major changes of pixel intensities among the pixels in the region. Nothing in this region should be detected. The fitness of composite operator on this region is defined as one minus the percentage of pixels detected in the region. If nothing is

Table 2.4. The performance on examples of IR and RGB color images. (f_{op} = fitness of the best composite operator, f_p = fitness of population, *: indicate fitness on training images, $f_{initial}$ = fitness in the initial generation, f_{final} = fitness in the final population)

	IR Image — People		Color Image — Car	
	f_{op}	f_p	f_{op}	f_p
$f_{initial}$	0.53	0.21	0.56	0.19
f_{final}	0.93 (0.85*, 0.83, 0.79, 0.85)	0.83	0.82 (0.75*, 0.73)	0.79
Ave. $f_{initial}$	0.60	0.21	0.43	0.18
Ave. f_{final}	0.89	0.73	0.72	0.70

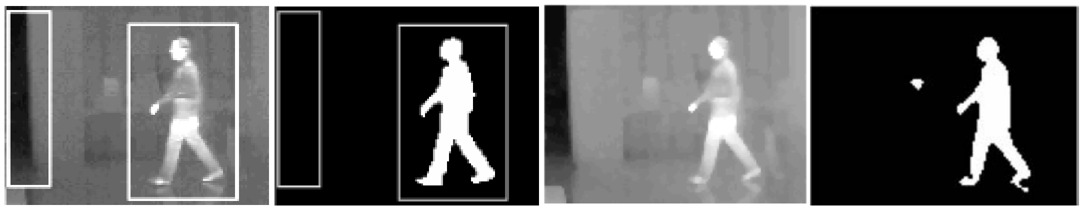

(a) person (b) ground truth (c) composite feature image (d) ROI extracted

Figure 2.21. Training IR image containing person.

detected, the fitness value is 1.0. Averaging the fitness values on the two training regions, we get the fitness during training. When the learned composite operator is applied to the whole training image, the fitness is computed as a measurement of the overlap between the ground truth and the extracted ROI, as we did in the previous experiments. Three testing IR images are shown in Figure 2.24(a), (d) and (g).

The generational GP is applied to synthesize composite operators for person detection and the results from the third run are reported. The fitness value of the best composite operator in the initial population is 0.53 and the population fitness value is 0.21. The fitness value of the best composite operator in the final population is 0.93 and the population fitness value is 0.83. Figure 2.21(c) shows the output image of the best composite operator on the whole training image and Figure 2.21(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.85. The best composite operator has 20 nodes and its depth is 9. It has five leaf nodes and is shown in Figure 2.22. Figure 2.23 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP proceeds.

```
(ADDC (LOG (MUL PFIM13 (MIN2
(MIN2 (MIN2 (ADDC (ADDC (LOG
PFIM3))) (ADDC (LOG PFIM3)))
(ADDC (LOG PFIM3))) (ADDC
(LOG PFIM13))))))
```

Figure 2.22. Learned composite operator tree in LISP notation.

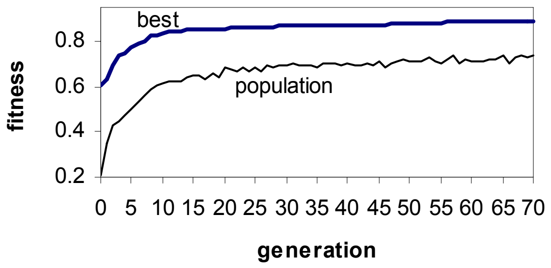


Figure 2.23. Fitness versus generation (person).

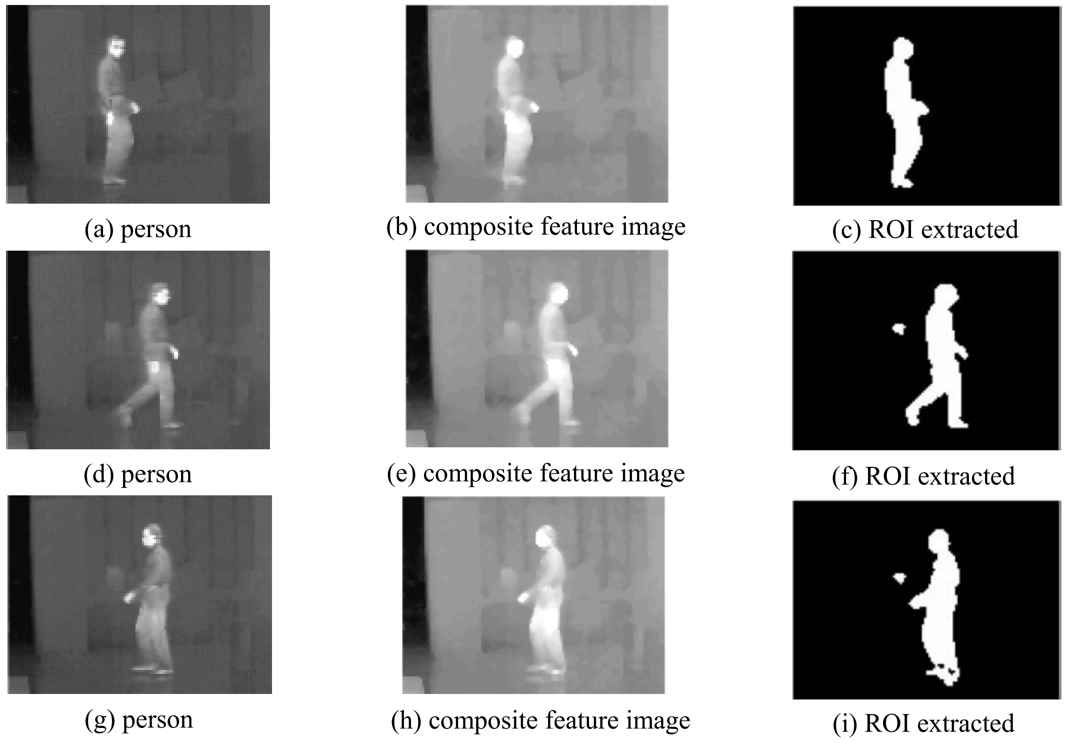

(a) person
(b) composite feature image
(c) ROI extracted
(d) person
(e) composite feature image
(f) ROI extracted
(g) person
(h) composite feature image
(i) ROI extracted

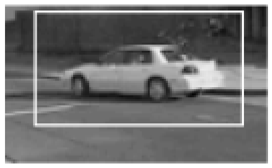
Figure 2.24. Testing IR images containing person.

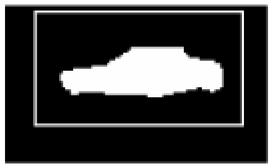
The composite operator is applied to the testing images shown in Figure 2.24. Figure 2.24(b), (e) and (h) show the output of the composite operator and Figure 2.24(c), (f) and (i) show the ROI extracted. Their fitness values are 0.83, 0.79 and 0.85 respectively.

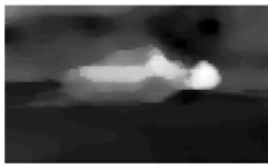
- **Car extraction in RGB color image:** GP is applied to learn features to detect car in RGB color images. Unlike previous experiments, the primitive feature images in this experiment are RED, GREEN and BLUE planes of RGB color image. Figure 2.25(a), (b) and (c) show the RED, GREEN and BLUE planes of the training image. The ground truth is shown in Figure 2.25(d). The training region is from (21, 3) to (91, 46).

(a) RED plane

(b) GREEN plane


(c) BLUE plane


(d) ground truth


(e) composite feature image

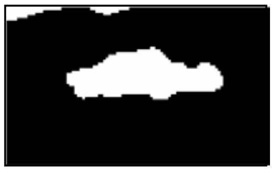

(f) ROI extracted

Figure 2.25. Training RGB color image containing car.

The steady-state GP is applied to synthesize composite operators for car detection and the results from the 4th run are reported. The fitness of the best composite operator in the initial population is 0.56 and the population fitness is 0.19. The fitness of the best composite operator in the final population is 0.82 and the population fitness is 0.79. Figure 2.25(e) shows the output image of the best composite operator on the whole training image and Figure 2.25(f) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.75. The best composite operator has 30 nodes and its depth is 18. It has six leaf nodes

```
(DIVC (MED (SUB PFB (MIN2
(ADD PFR (DIVC (DIVC (MULC
(DIVC (MULC PFB)))))) (ADDC
(DIVC (MULC (ADDC (MIN2 PFB
(MIN2 PFB (ADDC (DIVC (MULC
(ADDC (DIVC (MULC (DIVC
(MULC PFR))))))))))))))))))
```

Figure 2.26. Learned composite operator tree in LISP notation.

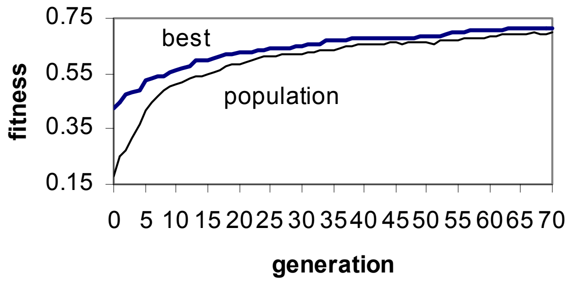
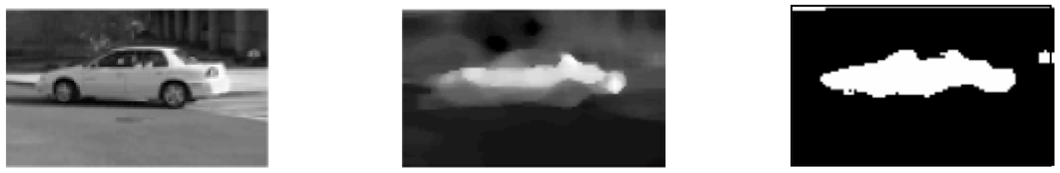


Figure 2.27. Fitness versus generation (car).

with two of them containing RED plane and the others contain BLUE plane. It is shown in Figure 2.26, where PFR means RED plane and PFB means BLUE plane. Figure 2.27 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP proceeds.

The composite operator is applied to the testing image whose RED plane is shown in Figure 2.28(a). Figure 2.28(b) shows the output of the composite operator and Figure 2.28(c) shows the ROI extracted. The fitness value of extracted ROI is 0.73.



(a) RED plane (b) composite feature image (c) ROI extracted

Figure 2.28. Testing RGB color image containing car.

2.4.3 Comparison with Example Region Selection

In our previous work [7], genetic programming was applied on a whole training image, not on carefully selected regions of the training image, to learn composite operators for object detection. The genetic programming running on a whole image is called image GP and the genetic programming of this chapter is called region GP. The differences between the method presented here and that in [7] are:

1) Unlike [7] where image GP is used during training, region GP runs on carefully selected region(s) in this chapter to reduce the training time.
2) Only the first mutation type in subsection 2.3.2 and only the first seven primitive feature images are used in [7]. With more mutation types and more primitive feature images used, the diversity of the composite operator population can be further increased.

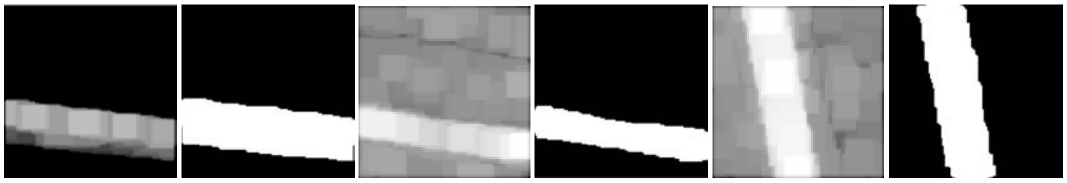
The experimental results on REAL SAR images in [7] are reported for the purpose of comparison. The parameters are: population size (100), the number of generations (100), the fitness threshold value (1.0), the crossover rate (0.6), the mutation rate (0.1), the maximum size (number of internal node) of composite operator (30), and the segmentation threshold (0). In each experiment, GP is invoked ten times with the same parameters. The experimental results from one run and the average performance of ten runs are reported in Table 2.5. We select the run in which GP finds the best composite operator among the composite operators found in all ten runs to report. The numbers in the parenthesis in the "f_{op}" columns are the fitness values of the best composite operators on the testing SAR images.

Road extraction: Figure 2.1(a) shows the training image and Figure 2.8(a), (d) show the testing images. The generational GP is used to generate a composite operator to extract the road. The fitness value of the best composite operator in the initial population is 0.47 and the population fitness value is 0.19. The fitness value of the best composite operator in the final population is 0.92 and the population fitness value is 0.89. Figure 2.29(a) shows the output image of the

Table 2.5. The performance of genetic programming on various examples of SAR images. (f_{op} = fitness of the best composite operator, f_p = fitness of population, *: indicate fitness on training images, $f_{initial}$ = fitness in the initial generation, f_{final} = fitness in the final population)

	Road		Lake		River		Field	
	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p
$f_{initial}$	0.47	0.19	0.65	0.42	0.43	0.21	0.62	0.44
f_{final}	0.92* (0.92, 0.89)	0.89	0.93* (0.92)	0.92	0.74* (0.84)	0.68	0.87* (0.68)	0.86
Ave. $f_{initial}$	0.47	0.18	0.73	0.39	0.37	0.11	0.65	0.41
Ave. f_{final}	0.81	0.76	0.92	0.87	0.68	0.58	0.84	0.77

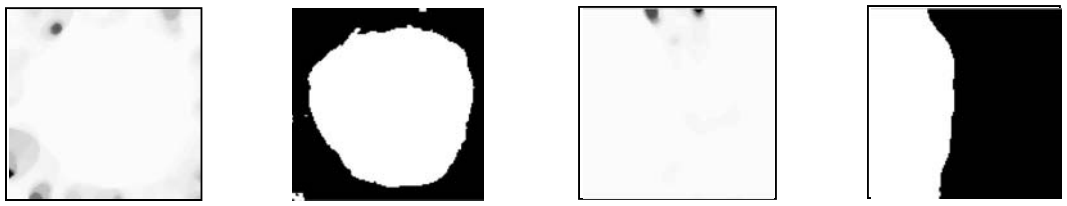
best composite operator in the final population and Figure 2.29(b) shows the extracted ROI. The composite operator obtained in the above training is applied to the two testing SAR images. Figure 2.29(c) and (d) show the output image of the composite operator and the ROI extracted from Figure 2.8(a). The fitness value of the extracted ROI is 0.92. Figure 2.29(e) and (f) show the output image of the composite operator and the ROI extracted from Figure 2.8(d). The fitness value of the extracted ROI is 0.89.



(a) composite feature image (b) ROI extracted from Figure 2.1(a) (c) composite feature image (d) ROI extracted from Figure 2.8(a) (e) composite feature image (f) ROI extracted from Figure 2.8(d)

Figure 2.29. Results on SAR images containing road.

Lake extraction: Figure 2.9(a) shows the training image and Figure 2.10(a) shows the testing image. The steady-state GP is used to generate composite operators. The fitness value of the best composite operator in the initial population is 0.65 and the population fitness value is 0.42. The fitness value of the best composite operator in the final population is 0.93 and the population fitness value is 0.92. Figure 2.30(a) shows the output image of the best composite operator in the final population and Figure 2.30(b) shows the extracted ROI. The



(a) composite feature image

(b) ROI extracted from Figure 2.9(a)

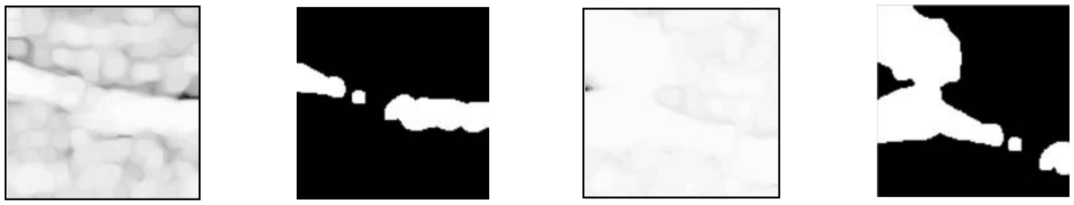
(c) composite feature image

(d) ROI extracted from Figure 2.10(a)

Figure 2.30. Results on SAR images containing lake.

composite operator is applied to the testing SAR image. Figure 2.30(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.92. In Figure 2.30(a) and (c), pixels in the small dark regions have very low pixel values (negative values with very large absolute value), making most pixels of the images appear bright, although some of these pixels have negative pixel values.

- **River extraction:** Figure 2.11(a) shows the training image and Figure 2.14(a) shows the testing image. The steady-state GP is used to generate the composite operator. The fitness value of the best composite operator in the initial population is 0.43 and the population fitness value is 0.21. The fitness value of the best composite operator in the final population is 0.74 and the population fitness value is 0.68. Figure 2.31(a) shows the output image of the best composite operator in the final population and Figure 2.31(b) shows the extracted ROI. The composite operator is applied to the testing image. Figure 2.31(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.84. Like Figure 2.30(c), pixels in the small dark region have very low pixel values (negative values with very large absolute value), thus, making many pixels with negative pixel values appear bright.



(a) composite feature image

(b) ROI extracted from Figure 2.11(a)

(c) composite feature image

(d) ROI extracted from Figure 2.14(a)

Figure 2.31. Results on SAR images containing river.

- **Field extraction:** Figure 2.15(a) shows the training image and Figure 2.16(a) shows the testing image. The generational GP was used to generate composite operators. The fitness value of the best composite operator in the initial population is 0.62 and the population fitness value is 0.44. The fitness value of the best composite operator in the final population is 0.87 and the population

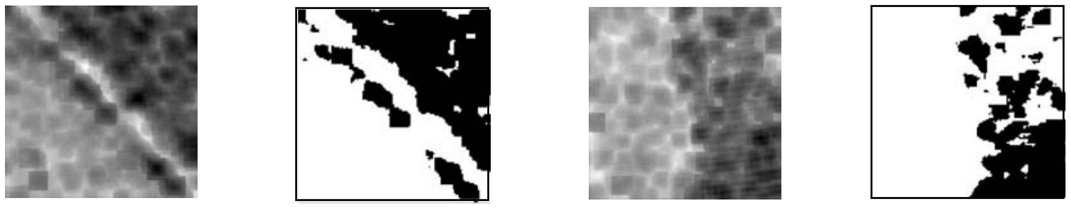

(a) composite feature image (b) ROI extracted from Figure 2.15(a) (c) composite feature image (d) ROI extracted from Figure 2.16(a)

Figure 2.32. Results on SAR images containing field.

fitness value is 0.86. Figure 2.32(a) shows the output image of the best composite operator in the final population and Figure 2.32(b) shows the extracted ROI. The composite operator is applied to the testing image. Figure 2.32(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.68.

From Tables 2.3 and 2.5 and associated figures, it can be seen that if the training regions are carefully selected and represent the characteristics of training images, the composite operators learned by GP running on training regions are effective in extracting the ROIs containing objects and their performances are comparable to the performances of composite operators learned by GP running on the whole training images. By running on the selected regions, the training time is greatly reduced. Table 2.6 shows the average running time of GP running on selected regions (Region GP) and GP running on the whole training images (Image GP) over all ten runs and the time is measured in second. Since the number of generation in [7] is 100 and the number of generation in this chapter is 70, the running time of "Image GP" in Table 2.6 is the actually running time of "Image GP" times 0.7. It can be seen that the training time using selected training region(s) is much shorter than that using a whole image.

Table 2.6. Average running time of Region GP and Image GP.

	Road	Lake	River	Field
Region GP	6915	2577	7951	3606
Image GP	23608	9120	66476	21485

2.4.4 Comparison with a Simple ROI Extraction Algorithm

To perform object detection, composite operators are used to extract ROIs that contain objects. In order to show the effectiveness of composite operators in ROI extraction, they are compared with a simple ROI extraction algorithm. The simple ROI extraction algorithm uses a threshold value to segment the image into foreground and background. The region consisting of pixels with value greater than the threshold value is called bright region and its complement is called dark region. If the bright region has a higher fitness than the dark region, the bright region is the foreground. Otherwise, the dark region is the foreground. The foreground is the ROI extracted by this simple algorithm. It is obvious that the threshold value plays a vital role in the ROI extraction and selecting an appropriate threshold value is the key to the success of the simple ROI extraction algorithm.

The performance of composite operators is compared with the simple ROI extraction algorithm when the best threshold value is used. To find the best threshold value, every possible threshold value is tried by the algorithm and its performance is recorded.

- **The Simple ROI Extraction Algorithm**

```
1. find the maximum and minimum pixel values of the image.
2. if the maximum pixel value is greater than 1000
3.     normalize the pixel values into the range of 0 to 1000. The pixel values
       are changed according to the following equation.
         new_pixval = (org_pixval – min_pixval) / (max_pixval – min_pixval) *
       1000
       where new_pixval and org_pixval are the new and original pixel values,
       respectively and min_pixval and max_pixval are the minimum and maxi-
       mum pixel values in the original image. After normalization, the minimum
       and maximum pixel values are 0 and 1000, respectively.
   else
       do not normalize the image.
   endif
4. each pixel value between the minimum and maximum pixel values is used as
   the threshold value and its performance in ROI extraction is recorded.
5. select the best threshold value and output its corresponding ROI.
```

Figure 2.33 shows the ROIs extracted by the simple ROI extraction algorithm corresponding to the best threshold value. The fitness values of the extracted ROIs and their corresponding threshold values are shown in Table 2.7.

Table 2.7. Fitness values of the extracted ROIs and the corresponding threshold values.

	Figure 2.33 (a)	Figure 2.33 (b)	Figure 2.33 (c)	Figure 2.33 (d)
Fitness	0.39	0.45	0.50	0.72
Threshold	24	32	29	36

	Figure 2.33 (e)	Figure 2.33 (f)	Figure 2.33 (g)	Figure 2.33 (h)
Fitness	0.81	0.31	0.55	0.63
Threshold	24	34	38	112

	Figure 2.33 (i)	Figure 2.33 (j)	Figure 2.33 (k)	Figure 2.33 (l)
Fitness	0.53	0.62	0.58	0.82
Threshold	128	129	107	95

	Figure 2.33 (m)	Figure 2.33 (n)	Figure 2.33 (o)	Figure 2.33 (p)	Figure 2.33 (q)
Fitness	0.83	0.79	0.84	0.38	0.34
Threshold	94	95	95	113	126

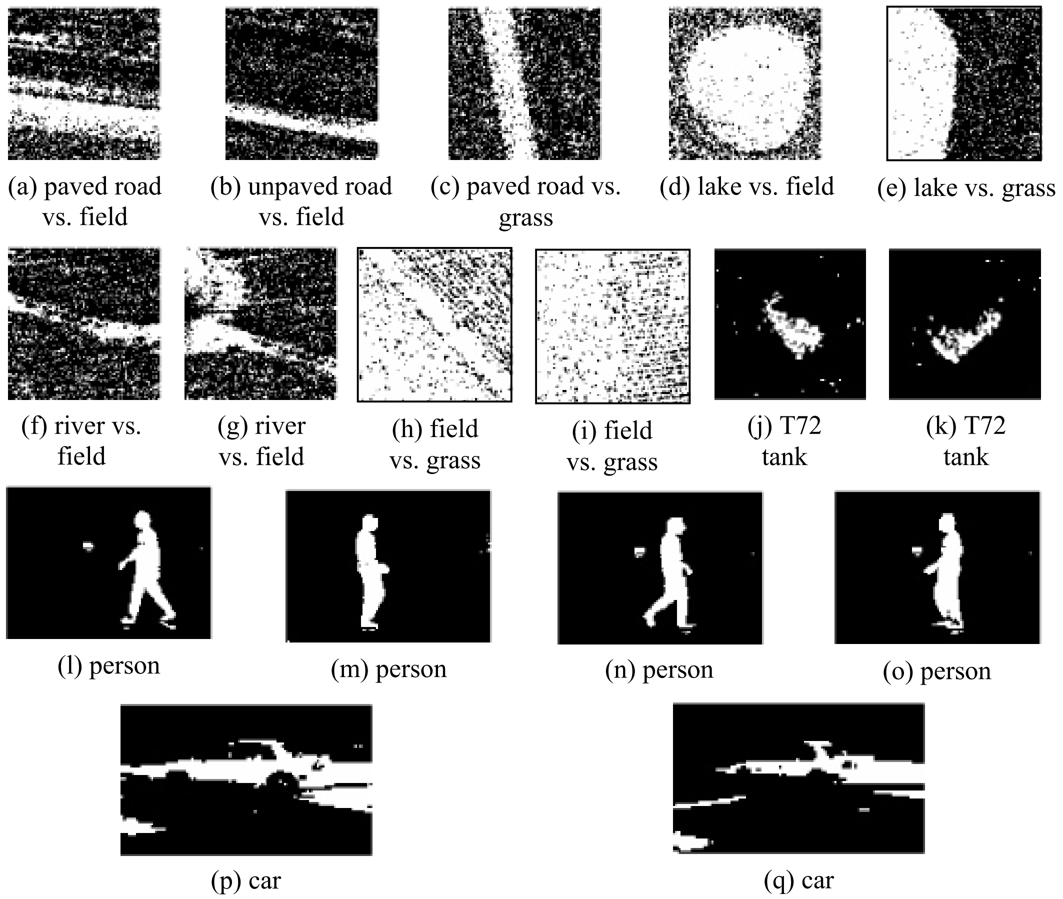


Figure 2.33. ROI extracted by the simple ROI extraction algorithm.

To extract ROIs from SAR and IR images, the original SAR and IR images are used by the simple ROI extraction algorithm; to extract ROIs from RGB color images, the color images are first converted into gray intensity images and the simple ROI extraction algorithm operates on the converted gray images. From Figure 2.33 and Table 2.7, it is obvious that the composite operators learned by

Table 2.8. Average running time of the composite operators and the simple ROI extraction algorithm.

	Road	Lake	River	Field	Tank	Person	Car
Composite operator	5	15	33	8	3	1	2
Simple ROI exaction algorithm	38	25.5	68.5	37.5	26	4.8	5.5

GP are more effective in ROI extraction. Actually, its performance is better than the best performance of the simple ROI extraction algorithm. Table 2.8 shows the average running time of the composite operators and the simple ROI extraction algorithm in extracting ROIs from training and testing images. The time is measured in seconds. From Table 2.8, it is obvious that the composite operators are more efficient.

2.5 Summary

In this chapter, the efficacy of genetic programming in synthesizing composite operators and composite features for object detection is investigated. The experimental results show that the primitive operators and primitive features defined are effective. GP can synthesize effective composite operators for object detection by running on the carefully selected training regions of a training image and the synthesized composite operators can be applied to the whole training image and other similar testing images. No significant difference is found between generational genetic programming and steady-state genetic programming. GP has a well-known code bloat problem. Controlling code bloat due to the limited computational resources inevitably restricts the search efficiency of GP. How to reach the balance point between these two conflicting factors is critical in the implementation of GP. In chapter 3, this problem will be addressed by designing a new fitness function based on minimum description length (MDL) principle to incorporate the size of a composite operator into the evaluation process. Also, this work will be extended by discovering features within the regions of interest for automated object recognition in chapters 4 and 5.

Chapter 3

Improving Efficiency of Genetic Programming for Object Detection

In chapter 2, the efficacy of genetic programming in learning composite features for object detection is studied. The motivation for using GP is to overcome the human experts' limitation of considering only a very limited number of conventional combinations of primitive features. Chapter 2 shows that GP is an effective way of synthesizing composite features from primitive ones for object detection. However, genetic programming is very computationally expensive. In the traditional GP (also called *normal GP*) used in chapter 2, crossover and mutation locations are randomly selected, leading to the disruption of the effective components (subtree in this report) of composite operators, especially at the later stage of the GP search. This greatly reduces the efficiency of GP. To improve the efficiency, it is very important for GP to identify and keep the effective components of composite operators. In this chapter, smart crossover and smart mutation are proposed to smartly choose crossover and mutation points to prevent effective components of a composite operator from being disrupted. Also, a public library is established to save the effective components of composite operators for later reuse. Finally, a fitness function based on the minimum description length (MDL) principle is designed to incorporate the size of a composite operator into the fitness evaluation to address the well-known code bloat problem of GP without imposing severe restrictions on the GP search. The GP with smart crossover, smart mutation and MDL-based fitness function is called *smart GP*.

3.1 Motivation

Crossover and mutation are two major mechanisms employed by GP to search the composite operator space. As GP proceeds, effective components are generated. The power of crossover lies in the fact that by swapping sub-trees between two effective composite operators (parents), the effective components (sub-trees) in these two parents can be assembled together into child composite operators (offspring) and the new offspring may be better than both parents. However, although crossover can assemble good components to yield better offspring, it is also a destructive force in the sense that it can disrupt good components due to the random selection of crossover points. When the search begins, since the initial population is randomly generated, it is unlikely that a composite operator contains good, especially large good, components and the probability of crossover breaking up a good component is small. At this time, crossover is a constructive force and the fitness of a composite operator is increased. As search proceeds, small good components are generated and assembled into larger and larger good components. When more and more composite operators contain large good components to achieve high fitness, the good component accounts for a large portion of a composite operator and the composite operator becomes more and more fragile because the good components are more prone to being broken up by subsequent crossover due to the random selection of crossover point. The crossover can damage the fitness of a composite operator in ways other than disrupting good components. Sometimes, a good component is moved into an inhospitable context, that is, the crossover inserts a good component into a composite operator that does not use the good component in any useful way or other nodes of the composite operator cancel out the effect of the good component. According to [2], crossover has an overwhelmingly negative effect on the fitness of the offspring from crossover, especially in the later stage of GP search. Mutation is introduced to maintain the diversity of population; since a serious weakness of evolutionary algorithms is that the population recombined repeatedly will develop uniformity sooner or later [2]. However, in the later stage of GP search when more and more composite operators contain large good components, the random selection of mutation point leads to the high probability of disrupting good components and makes mutation a destructive force. When both crossover and mutation become negative factors in the GP search, it is very unlikely that better composite operators will be generated. To improve the efficiency and effectiveness of GP, it is highly desired that good components can be identified and kept from destructive crossover and mutation operations and stored in a public library for later reuse. These components are treated as atomic terminals and are directly inserted into composite operators as a whole when the mutations are performed or during initialization.

GP has a well-known code bloat problem in which the sizes of individuals become larger and larger. In this report, the individuals are composite operators represented by binary trees. In normal GP, crossover is performed by swapping the sub-trees rooted at the nodes randomly selected as crossover points. It is easy to see that the size of one offspring (i.e., the number of nodes in the binary tree representing the offspring) may be greater than both parents if crossover is performed in this simple way. If we do not control the sizes of composite operators, they will become larger and larger as GP proceeds. This problem must be addressed, since when the size becomes too large, it takes a long time to execute a composite operator, greatly reducing the speed of GP. Large-size composite operators may overfit training data by approximating the noises in images. Although the result on the training image is very good, the performance on unseen testing images may be bad. Also, large composite operators take up a lot of computer memory. Due to the limited computer resources and for the running speed of GP, usually in normal GP, a limit on the size of composite operators is established when performing crossover or mutation. If the size of an offspring exceeds the maximum size allowed, the crossover operation is performed again until the sizes of both offspring are within the limit. Although this simple method prevents the code bloat, the hard size limit may greatly restrict the search performed by GP, since after randomly selecting a crossover point in one composite operator, GP cannot select some nodes of the other composite operator as crossover point in order to guarantee that both offspring won't exceed the size limit. However, restricting the search greatly reduces the efficiency of GP, making it less likely to find good composite operators. One may suggest that after two composite operators are selected, GP performs crossover twice and each time keeps the offspring of smaller size. This method can enforce the size limit and prevent the size of offspring composite operators from growing large, but GP now only searches the space of these small composite operators. With a small number of nodes, a composite operator may not capture the characteristics of the objects to be detected. How to avoid restricting the GP search while at the same time preventing code bloat is crucial to the success of GP and is still a subject of research. The key is to find a balance point between these two conflicting factors. In this chapter, a fitness function is designed based on minimum description length (MDL) principle [12] to take the size of a composite operator into the fitness evaluation process. According to MDL principle, large composite operators effective on training regions may not have good fitness. With the new MDL-based fitness function, we can take off the restriction on the selection of crossover points while preventing the composite operator from growing too large, since these large composite operators don't have high fitness and will be culled out by selection.

3.2 Related Research

To improve the efficiency of GP, Tackett [13] devises a method called brood recombination to reduce the destructive effect of crossover. In this method, when crossover is performed, many offspring are generated from two parents and only the best two offspring are kept. D'haeseleer [14] devises strong context preserving crossover (SCPC) to preserve the context. SCPC only permits crossover between nodes that occupied exactly the same position in the two parents. He finds modest improvement in results by mixing regular crossover and SCPC. Smith [15] proposes a conjugation operator for GP to transfer genetic information from one individual to another. In his conjugation method, the parent with higher fitness becomes the donor and the other with lower fitness becomes the recipient. The conjugation operator is different from crossover and it simulates one of the ways in which individuals exchange genetic materials in nature. Ito et al. [16] propose a depth-dependent crossover for GP in which the depth selection ratio is varied according to the depth of a node. A node closer to the root node of the tree has a better chance of being selected as a crossover point to lower the chance of disrupting small good components near leaves. Their experimental results show the superiority of the depth-dependent crossover to the random crossover in which crossover points are randomly selected. Bhanu and Lin [17] propose smart crossover and mutation operators to identify and keep the good components of composite operators. Their initial experiments show that with smart GP operators, GP can search the composite operator space more efficiently.

Unlike the work of Ito [16] that used only the syntax of a tree (the depth of a node), the smart crossover and smart mutation proposed in this chapter evaluate the performance of each node to determine the interactions among them and use the fitness values of the nodes to determine crossover and mutation points. Also, unlike my previous work [17], a public library is introduced to keep the good components for later reuse and more types of mutations are added to increase the population diversity. Nine more primitive feature images are included to build composite operators. To reduce the training time, the training in this chapter is performed on the selected regions of training images, not the whole images as in the previous work. More importantly, a new MDL-based fitness function is designed to reach a balance point between conflicting factors of code bloat and less restriction on the GP search.

3.3 MDL-based fitness function

Fitness function plays an important role in the GP search. It determines the direction and efficiency of GP search. Driven by a good fitness function, GP can generate effective composite operators more quickly. One important question in the synthesis of composite operator is to determine the appropriate size of composite operators to prevent overfitting while at the same time capture the characteristics of objects. With little knowledge on the composite operator space and the object characteristics, it is difficult to answer such questions and this is one of the reasons GP is applied to explore this vast space. It is shown in chapter 2 that the composite operator space is very large. In order to find effective composite operators, GP must search actively. To address the well-known code bloat problem and prevent severe restriction on the GP search, an MDL-based fitness function is designed to incorporate the composite operator size into the fitness evaluation process. The fitness of a composite operator is defined as the sum of description length of the composite operator and the description of the segmented training regions with respect to this composite operator as a predictor for the label (object or background) of each pixel in the training regions. Here, both lengths are measured in bits and the details of the coding techniques are relevant. The trade-off between the simplicity and complexity of a composite operator is that if the size of a composite operator is too small, it may not capture the characteristics of the objects to be detected, on the other hand, if the size is too large, the composite operator may overfit the training image, thus performing badly on unseen testing images. With the MDL-based fitness function, the composite operator with the minimum combined description length of both the operator itself and image-to-operator error is the best composite operator and may perform best on unseen testing images.

Based on the minimum description length principle, the following fitness function is proposed for GP to maximize:

$$F(CO_i) = - (r\times\log (N_{po}) \times Size(CO_i) + (1 - r) \times (n_o + n_b) \times(\log(W_{im})+\log(H_{im})) \quad (3.1)$$

where CO_i is the *i*th composite operator in the population, N_{po} is the number of primitive operators available for GP to synthesize composite operators, $Size(CO_i)$ is the size of the composite operator which is the number of nodes in the binary tree representing it, n_o and n_b are the number of object and background pixels misclassified and W_{im} and H_{im} are the width and height of the training image and r is a parameter determining the relative importance of the composite operator size and the detection rate, which is 0.7 in this chapter.

We now give a brief explanation of this fitness function. Suppose a sender and a receiver both have the training image and the training regions and they agree in advance that composite operators can be use to locate the object in an image, that is, to determine the label (object or background) of each pixel in training regions. But only the sender knows the ground truth (the label of each pixel). Now, the sender wants to tell the receiver which pixels belong to the object and which pixels belong to the background. One simple approach to do this is to send a bit sequence of n (n is the number of pixels in the training regions) bits where 1 represents an object pixel and 0 represents a background pixel, provided that both the sender and receiver know the order of the training regions and they agree that the pixels are scanned in the top-to-bottom and left-to-right fashion. However, n is usually very large, thus the communication burden is heavy. To reduce the number of bits to be transmitted, the sender can send the composite operator to the receiver. Then the receiver applies the composite operator on the training regions to get segmented training regions. When sending the composite operator, the sender can send its nodes in a pre-traversal order. Given N_{po} primitive operators (include primitive feature images), $\log(N_{po})$ is needed to encode each node. Thus the cost of sending a composite operator is $\log (N_{po}) \times Size(CO_i)$. However, some pixels may be misclassified by the composite operator. In order for the receiver to get the truth, the sender needs to tell the receiver which pixels are misclassified. Each pixel is represented by its coordinate in the image. If the width and height of an image are W_{im} and H_{im} respectively, then $\log(W_{im})+\log(H_{im})$ bits are needed to encode each pixel. Thus the cost of sending the misclassified pixels is $(n_o + n_b)$

×(log(W_{im})+log(H_{im})). If the composite operators are very effective and its size is not too large, then only few pixels are misclassified and the number of bits to send is much smaller than n.

In chapter 2, the fitness function is defined as *n(GÇG¢) / n(G È G¢)*, where *G* and *G¢* are foregrounds in the ground truth image and the resultant image of a composite operator respectively and *n(X)* denotes the number of pixels in the intersection of region *X* and the training regions. It measures how the ground truth and the detection results are overlapped. In this chapter, this measure is called the goodness of a composite operator. It is not used to drive GP, but only used to measure the effectiveness of a composite operator.

3.4 Technical Approach

- **Primitive feature images and operators:** The primitive feature images and primitive operators are the same as those used in chapter 2. There are 16 primitive feature images: the original image (0), mean (1–3), deviation (4–6), maximum (7–9), minimum (10–12) and median (13–15) images obtained by applying templates of sizes 3×3, 5×5 and 7×7. 17 primitive operators are ADD, SUB, MUL, DIV, MAX2, MIN2, ADDC, SUBC, MULC, DIVC, SQRT, LOG, MAX, MIN, MED, MEAN and STDV.

- **Parameters and termination:** The key parameters are the population size M, the number of generation N, the crossover rate, the mutation rate and the goodness threshold. The GP stops whenever it finishes the pre-specified number of generations or whenever the best composite operator in the population has goodness value greater than the goodness threshold.

- **Selection, crossover and mutation:** The selection operation selects composite operators from the current population to let them survive into next generation. As in chapter 2, tournament selection is used.

In normal GP, crossover and mutation points are selected at random. Due to the random selection, crossover and mutation become destructive at the later stage of GP search when composite operators contain large effective components. To avoid the above problem, smart crossover and smart mutation are proposed to identify and keep the effective components. In smart GP, the output image of each node is evaluated and its fitness value is recorded. The fitness of an edge is defined as the fitness difference between the parent node and the child node linked by the edge. An *Edge* is classified as good edge if its fitness is positive. Otherwise, it is a bad edge.

In the smart crossover, all the bad edges are identified and one of them is selected by random selection or roulette selection (based on the fitness of the bad edges) invoked with equal probability. The child node of the selected bad edge is the crossover point and the subtree rooted at the crossover points are swapped between parents. If a composite operator has no bad edge, the crossover point is randomly selected.

A *public library* is established to store good components for later reuse by smart mutation. The larger the library, the more effective components can be kept for later reuse, but the likelihood of each effective component being reused is reduced. In this chapter, the public library stores 100 good components.

In the smart mutation, mutation point is the parent or child node of a bad edge or a bad node whose goodness value is below the average goodness value of all the nodes. The mutation point is selected among those qualified nodes at random. There are four smart mutations invoked with equal probability:

(a) select the parent node of a bad edge as mutation point. If the parent node has only one child, the parent is deleted and the child node is linked to the grand parent node (parent node of the parent node), if no grand parent node exists, the child becomes the root node; if the parent node has two children, the parent node and the sub-tree rooted at the child with smaller fitness value are deleted and the other child is linked directly to the grand parent node, if no grand parent node exists, the child becomes the root node;

(b) select the parent a node of a bad edge as mutation point and replace the primitive operator stored in the node with another primitive operator with the same number of input as the replaced one;

(c) select two subtrees whose roots are the child nodes of two bad edges within the composite operator and swap them. Of course, neither of the two sub trees can be a sub-tree of the other;

(d) select a bad node as mutation point. Replace the subtree rooted at the node with another randomly generated tree or with an effective component randomly selected from the public library.

The first two mutations delete a node that cancel the effect of its child or children; the third mutation moves two components away from unfriendly contexts that cancel their effects and inserts them into new contexts; the fourth mutation deletes a bad component and replace it with a new component or a good one stored in the public library.

ε-greedy policy is used to determine whether a smart operator (smart crossover or mutation) or a random operator (random crossover or mutation) is used. The smart operator and random operator are invoked with probability ε and $1 - \varepsilon$, respectively. In this chapter, ε is a variable and can be adjusted by the following formula:

$$\varepsilon = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) \times Good_{popu} \qquad (3.2)$$

where ε_{min} is 0.5 and ε_{max} is 0.9, $Good_{popu}$ is population goodness (the average goodness of the composite operators in the population). The reason for the using of random operators is that smart operators bias the selection of crossover and mutation points. They avoid disrupting effective components, but at the same time they restrict the GP search. According to our experiments, restricting the search reduces the efficiency of GP. At the beginning when the population is initialized, few composite operators contain effective components. At this time, GP should search actively to generate effective components and assemble them together. It is harmful to apply smart operators at the early stage of GP search since they restrict the search. Only after some time when the effective components are gathered in composite operators, smart operators should be applied to identify the effective components to avoid disrupting them and keep them in a public library for later reuse. So, in this report, smart operators are not used in the first 20 generations. In the last 50 generations, smart operators are applied with higher and higher probability as the population goodness becomes larger and larger.

• **Generational GP and smart GP:** As in chapter 2, generational genetic programming and steady-state genetic programming are used to synthesize composite operators. The difference is that in smart GP, MDL-based fitness function is used, smart GP crossover and smart mutation are invoked with probability determined by ε-greedy policy and a public library is set up to store the effective components of composite operators.

Generational Genetic Programming:

```
0.  randomly generate populations of size M and evaluate each composite opera-
    tor in P.
1.  for gen = 1 to N do   // N is the number of generation
2.      keep the best composite operator in P.
3.      perform crossover on the composite operators in P until crossover rate is
        satisfied and keep all the offspring from crossover. if gen < 20 use random
        crossover only; otherwise smart crossover and random crossover are in-
        voked according to e-greedy policy.
4.      perform mutation on the composite operators in P and the offspring from
        crossover with the probability of mutation rate. if gen < 20 use random
        mutation only; otherwise smart mutation and random mutation are invoked
        according to e-greedy policy.
5.      perform selection on P to select some composite operators. The number of
        selected composite operators must be M minus the number of composite
        operators from crossover.
6.      combine the composite operators from crossover with those from P to get a
        new population P'.
7.      evaluate offspring from crossover and the mutated composite operators.
```

```
8.      let the best composite operator from P replace the worst composite opera-
        tor in P' and let P = P'.
9.      update the value of e according to equation (3.2) and store good compo-
        nents of composite operators in the public library.
10.     if the goodness of the best composite operator in P is above the goodness
            threshold then
11.         stop.
        endif
12.     check each composite operator in P and use its best component to replace
        it.
    endfor // loop 1
```

Steady-state Genetic Programming:

```
0.  randomly generate population P of size M and evaluate composite operators
    in P.
1.  for gen = 1 to N do   // N is the number of generation
2.      keep the best composite operator in P.
3.      repeat
4.          select 2 composite operators from P based on their fitness values for
            crossover.
5.          select 2 composite operators with the lowest fitness values in P for re-
            placement.
6.          perform crossover operation and let the 2 offspring replace the 2 com-
            posite operators selected for replacement. if gen < 20 use random cross-
            over only; otherwise smart crossover and random crossover are invoked
            according to e-greedy policy.
7.          execute the 2 offspring and evaluate their fitness values.
8.      until crossover rate is met.
9.      perform mutation on each composite operator with probability of mutation
        rate. if gen < 20 use random mutation only; otherwise smart mutation and
        random mutation are invoked according to e-greedy policy.
10.     execute and evaluate mutated composite operators.
11.     after crossover and mutation, a new population P¢ is generated.
12.     let the best composite operator of population P replace the worst composite
        operator in P' and let P = P'.
13.     update the value of e according to equation (3.2) and store good compo-
        nents of composite operators in the public library.
14.     if the goodness of the best composite operator in P is above goodness
        threshold value then
15.         stop.
        endif
16.     check each composite operator in P and use its best component to replace
        it.
    endfor  // loop 1
```

3.5 Experiments

Experiments are performed with real SAR images of sizes 128×128 and 80×80 (tank images). To synthesize composite operators, GP is applied to a region (or regions) carefully selected from the training image to reduce the training time. The generated composite operator is then applied to the whole training image and other testing images. For the purpose of objective comparison, *normal GP* (GP with random crossover and random mutation) and *smart GP* (GP with smart crossover and smart mutation) are invoked ten times with the same parameters and training regions in each experiment and only the average performances are used in comparison. The results from the run in which GP finds the best composite operator among the best composite operators found in all ten runs are reported. The parameters are: population size (100), the number of generation (70), the goodness threshold value (1.0), the crossover rate (0.6), the mutation rate (0.05), and the segmentation threshold (0). The GP program ran on a Sun Ultra 2 workstation.

3.5.1 Road Extraction

The training image contains horizontal paved road and field, as shown in Figure 3.1(a); two testing images contain unpaved road vs. field and vertical paved road vs. grass, as shown in Figure 3.7(a) and 3.7(f), respectively. Two training regions locate from (5, 19) to (50, 119) and from (82, 48) to (126, 124). Figure 3.1(b) shows the ground truth. The white region corresponds to the road and only the portion of ground truth in the training regions is used in the fitness evaluation.

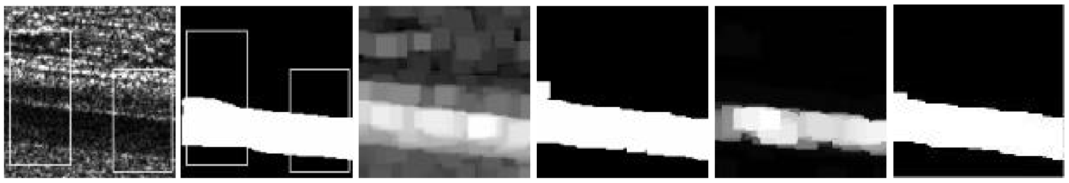

(a) paved road vs. field (b) ground truth (c) feature image (normal GP) (d) ROI extracted (normal GP) (e) feature image (smart GP) (f) ROI extracted (smart GP)

Figure 3.1. Training SAR image containing road.

The generational GP is used to synthesize a composite operator to detect the road. For normal GP, the goodness value of the best composite operator in the initial population is 0.60 and the goodness value of the best composite operator in the final population is 0.94. Figure 3.1(c) shows the output image of the best composite operator on the whole training image and Figure 3.1(d) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.90. For smart GP, the fitness and goodness of the best composite operator in the initial population are –2303.6 and 0.45. The corresponding values in the finial population are –325.4 and 0.94. Figure 3.1(e) shows the output image of the best composite operator on the whole training image and Figure 3.1(f) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.91.

The best composite operator has 18 nodes and its depth is 13. It has three leaf nodes all containing 7×7 median image, which contains less speckle noises due to the median filter's effectiveness in eliminating speckle noises. It is shown in Figure 3.2, where PFIM15 represents 7×7 median image. Compared to smart GP, the best composite operator from normal GP has 27 nodes and its depth is 16.

Figure 3.3 shows how the average fitness of the best composite operators and the average fitness of the populations over all 10 runs change as GP proceeds. Unlike in chapter 2 where the population fitness approaches the fitness of the best composite operator as GP proceeds, in Figure 3.3, population fitness is much lower than that of the best composite operator even at the end of GP search. It is reasonable, since the selection of crossover points is not restricted by a hard size limit on composite operators. The difference between the sizes of the composite operators in the population is large and so are their fitness values. The population fitness is not important since only the best composite operator is used in testing.

```
(MAX (MAX (MAX (MAX
(MAX (SUBC (MUL (DIVC
(ADDC (MAX (MAX (MAX
(ADDC PFIM15)))))) (DIV
PFIM15 (STDV PFIM15)))))))))
```

Figure 3.2. Learned composite operator tree in LISP notation.

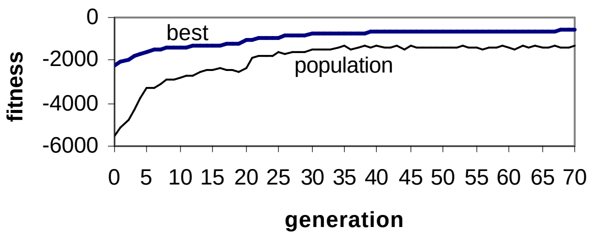


Figure 3.3. Fitness versus generation (road vs. field).

If GP finds one effective composite operator, the GP learning is successful. That's why we don't compare the population fitness between normal GP and smart GP. The large difference between the fitness of the best composite operator and that of the population indicates that the diversity of the population is maintained during GP search, which is very helpful in preventing premature convergence.

Ten best composite operators are obtained in the initial and final generations of ten runs, respectively. Figure 3.4 shows the utility of primitive operators and primitive feature images in the best composite operators of initial and final generations. To compute utility, we first compute the total number of each primitive operator and the total number of each primitive feature image in the 10 best composite operators, then divide them by the total number of internal nodes and the total number of leaf nodes of these 10 best composite operators, respectively. From Figure 3.4(b), it can be seen that MED operator has the most frequent occurrence in the best composite operators learned by GP.

Figure 3.5 shows the output image of each node of the best composite operator shown in Figure 3.2. The primitive operators in Figure 3.5 are connected by arrow. The operator at the tail of an arrow provides input to the operator at the head of the arrow. After segmenting the output image of a node, we get the ROI

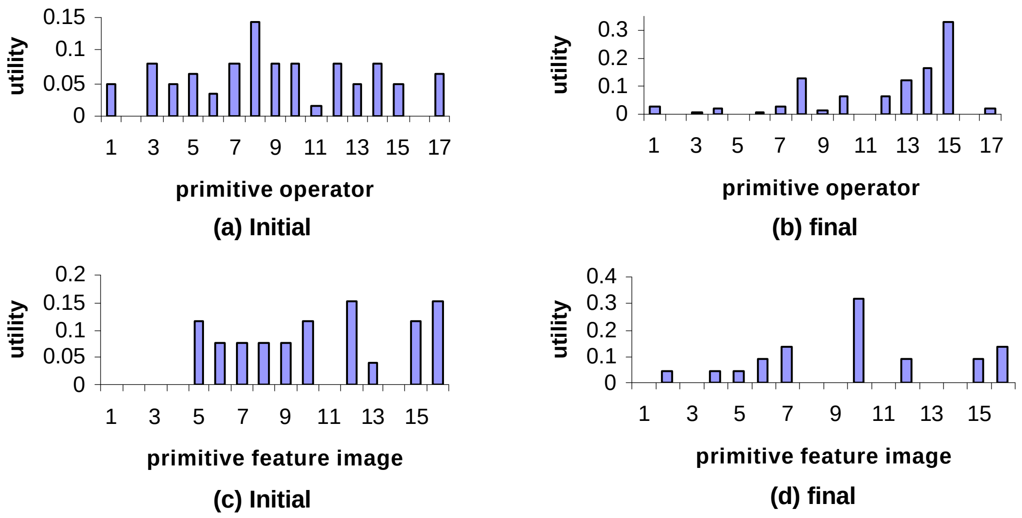


Figure 3.4. Utility of primitive operators and primitive feature images.

(shown as the white region) extracted by the corresponding subtree rooted at the node. The extracted ROIs and their fitness values are shown in Figure 3.6. If an output image has positive pixels only (for example, PFIM15 has positive pixels only), everything is extracted and the goodness is 0.25. From Figure 3.6, it can be seen that since the feature image from subtree (DIV PFIM15 (STDV PFIM15)) has no pixel with negative value, it does not affect the ROI extracted from the feature image output by its parent node MUL. This branch is a redundant code of the composite operator.

The composite operator obtained in the above training is applied to the other two real SAR images shown in Figure 3.7(a) and 3.7(f). Figure 3.7(b) and 3.7(g) show the output of the composite operator from normal GP and Figure 3.7(c) and Figure 3.7(h) show the regions extracted from Figure 3.7(a) and Figure 3.7(f), respectively. The goodness values of the extracted regions are 0.90 and 0.93. Figure 3.7(d) and 3.7(j) show the output of the composite operator from smart GP and Figure 3.7(e) and Figure 3.7(k) show the regions extracted from Figure 3.7(a) and Figure 3.7(f), respectively. The goodness values of the extracted regions are 0.91 and 0.93. The average running time of the best composite operators from normal GP on training and testing images is 5 seconds; the corresponding time of that from smart GP is 2.6 seconds.

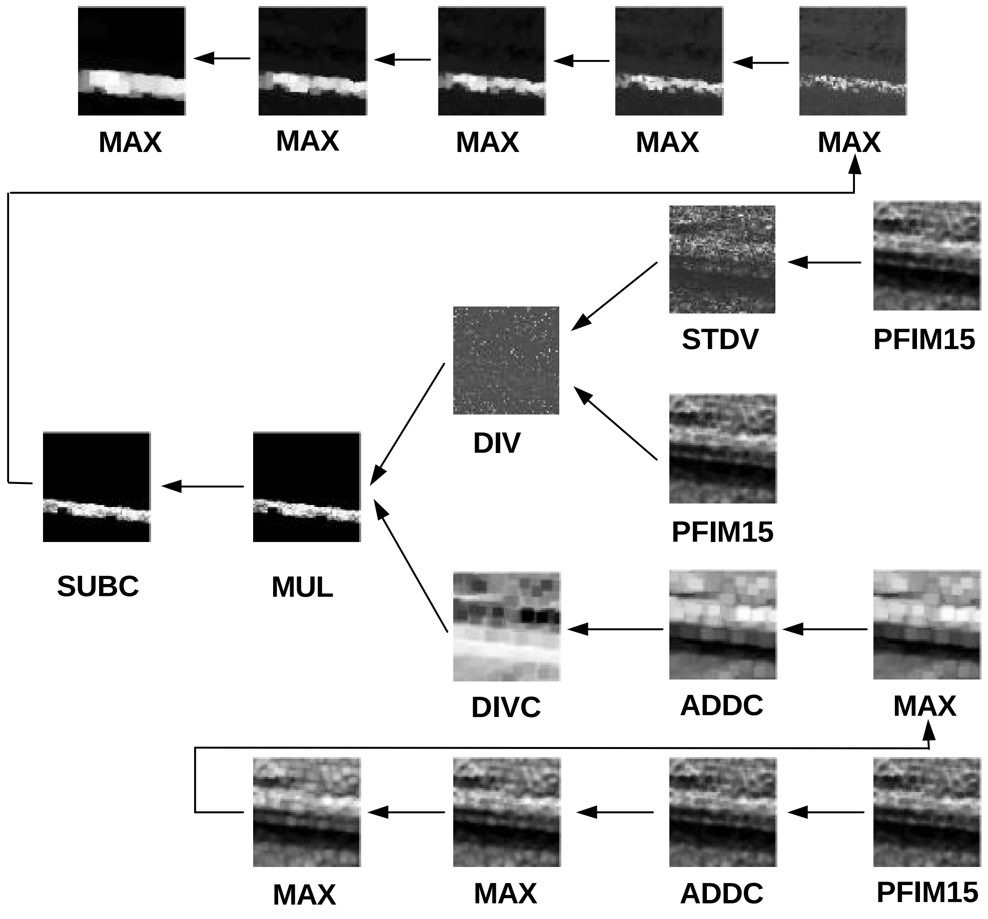


Figure 3.5. Feature images output by the nodes of the best composite operator from smart GP.

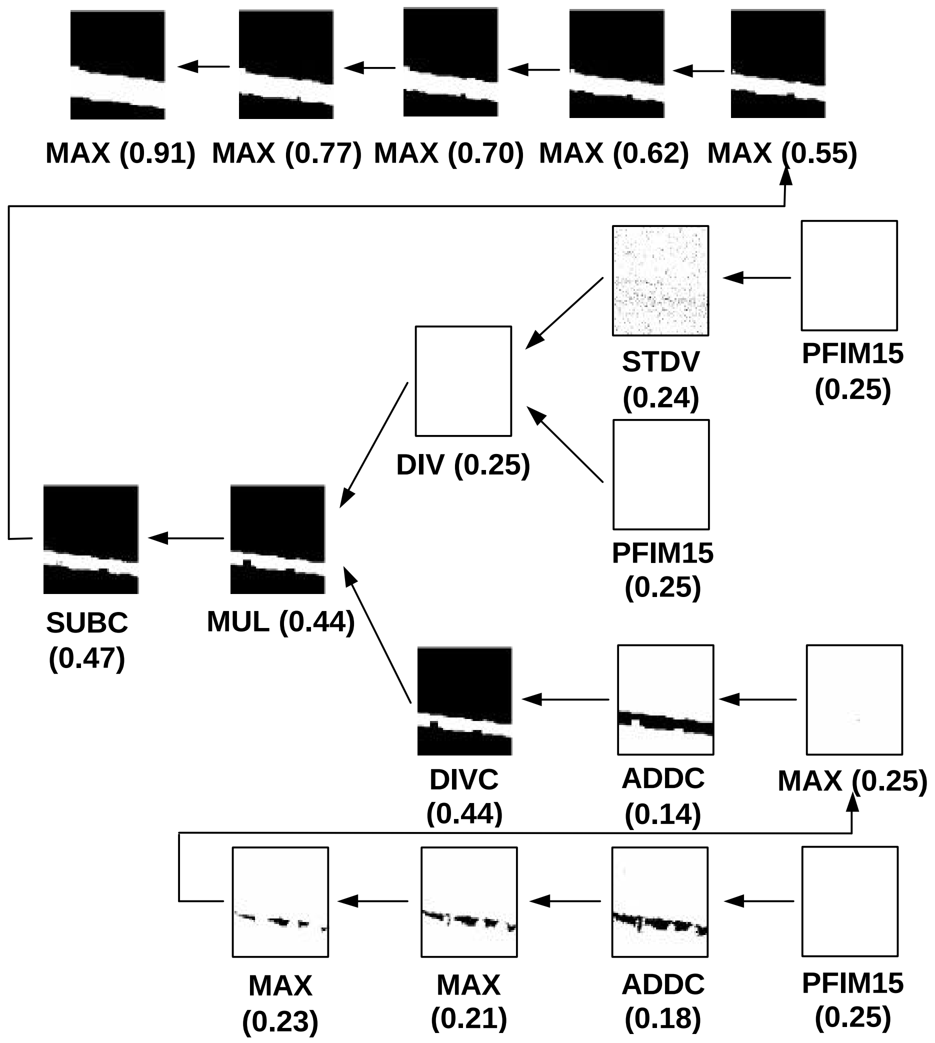


Figure 3.6. ROIs extracted from the output images of the nodes of the best composite operator from smart GP. (The fitness value is shown for the entire image)

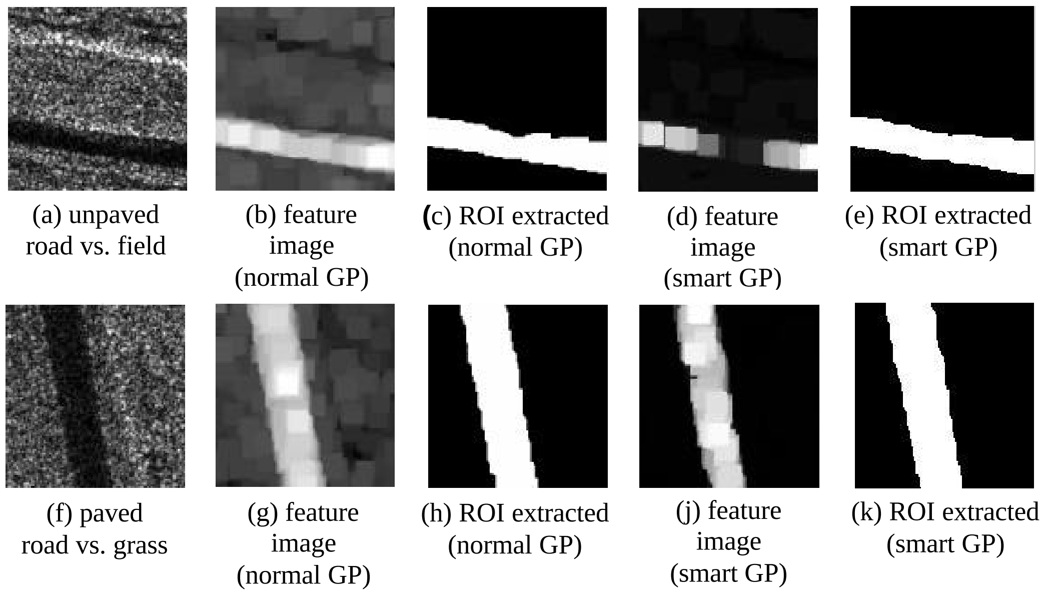

(a) unpaved road vs. field
(b) feature image (normal GP)
(c) ROI extracted (normal GP)
(d) feature image (smart GP)
(e) ROI extracted (smart GP)
(f) paved road vs. grass
(g) feature image (normal GP)
(h) ROI extracted (normal GP)
(j) feature image (smart GP)
(k) ROI extracted (smart GP)

Figure 3.7. Testing SAR images containing road.

3.5.2 Lake Extraction

Two SAR images contain lake. The training image, shown in Figure 3.8(a), contains a lake and field, and the testing image, shown in Figure 3.9(a) contains a lake and grass. The training region is from (85, 85) to (127, 127). Figure 3.8(b) shows the ground truth.

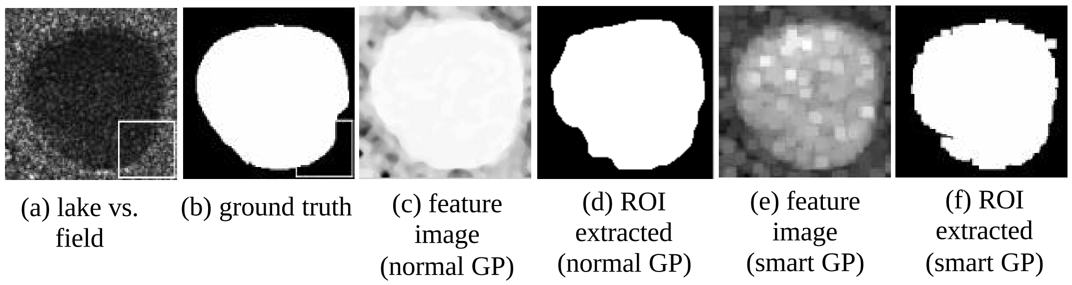

(a) lake vs. field
(b) ground truth
(c) feature image (normal GP)
(d) ROI extracted (normal GP)
(e) feature image (smart GP)
(f) ROI extracted (smart GP)

Figure 3.8. Training SAR image containing lake.

The steady-state GP is used to generate composite operators. For normal GP, the goodness value of the best composite operator in the initial population is 0.62 and the goodness value of the best composite operator in the final population is 0.99. Figure 3.8(c) shows the output image of the best composite operator on the whole training image and Figure 3.8(d) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.95. For smart GP, the fitness and goodness of the best composite operator in the initial population are –1585.5 and 0.55. The corresponding values in the finial population are –158.9 and 0.97. Figure 3.8(e) shows the output image of the best composite operator on the whole training image and Figure 3.8(f) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.94. The composite operator is applied to the testing image containing a lake and grass. Figure 3.9(b) shows the output of the composite operator from normal GP and Figure 3.9(c) shows the region extracted. The goodness value of the region is 0.97. Figure 3.9(d) shows the output of the composite operator from smart GP and Figure 3.9(e) shows the region extracted. The goodness value of the region is 0.98. The average running time of the best composite operators from normal GP on training and testing images is 15 seconds; the corresponding time of that from smart GP is 1 second. The sizes of the best composite operators from normal GP and smart GP are 28 and 13, respectively.

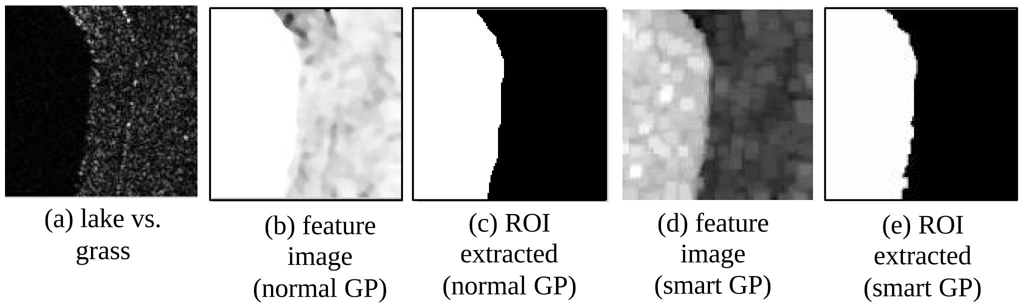

(a) lake vs. grass
(b) feature image (normal GP)
(c) ROI extracted (normal GP)
(d) feature image (smart GP)
(e) ROI extracted (smart GP)

Figure 3.9. Testing SAR image containing lake.

3.5.3 River Extraction

Two SAR images contain river and field. Figure 3.10(a) and 3.10(b) show the original training image and the ground truth provided by the user. The white region in Figure 3.10(b) corresponds to the river to be extracted. The training regions are from (68, 31) to (126, 103) and from (2, 8) to (28, 74). The testing SAR image is shown in Figure 3.13(a).

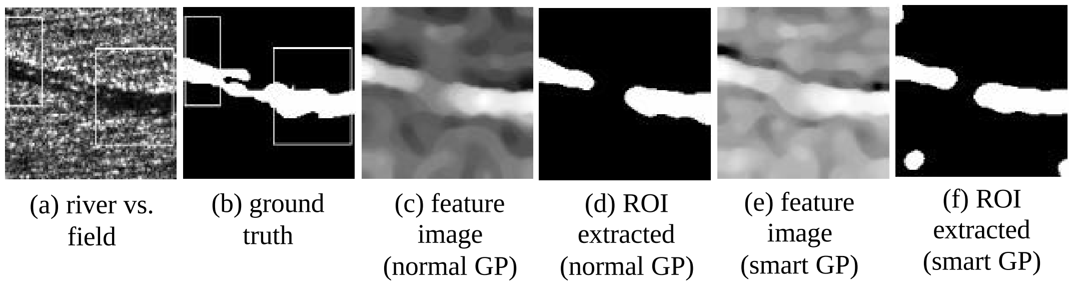

(a) river vs. field
(b) ground truth
(c) feature image (normal GP)
(d) ROI extracted (normal GP)
(e) feature image (smart GP)
(f) ROI extracted (smart GP)

Figure 3.10. Training SAR image containing river.

The steady-state GP is used to generate a composite operator. For normal GP, the goodness values of the best composite operator in the initial and final populations are 0.59 and 0.89, respectively. Figure 3.10(c) shows the output image of the best composite operator on the whole training image and Figure 3.10(d) shows the binary image after segmentation. The goodness of the extracted ROI is 0.72. For smart GP, the fitness and goodness of the best composite operator in the initial population are –2480.8 and 0.23. The corresponding values in the finial population are –404.6 and 0.90. Figure 3.10(e) shows the output image of the best composite operator on the whole training image and Figure 3.10(f) shows the binary image after segmentation. The goodness of the extracted ROI is 0.71. The best composite operator has 13 nodes and its depth is 12. It has one leaf node containing 3×3 mean image. Among 13 nodes, seven of them are MED operators effective in eliminating speckle noises. It is shown in Figure 3.11. Compared to smart GP, the best composite operator from normal GP has 30 nodes with depth 23. Figure 3.12 shows how the average fitness of the best composite operators and the average fitness of the populations over all 10 runs change as GP searches the composite operator space.

```
(DIVC (SUBC (SUBC (MED
(MED (MIN (MED (MED (MED
(MED (MED (SUBC
PFIM1))))))))))))
```

Figure 3.11. Learned composite operator tree in LISP notation.

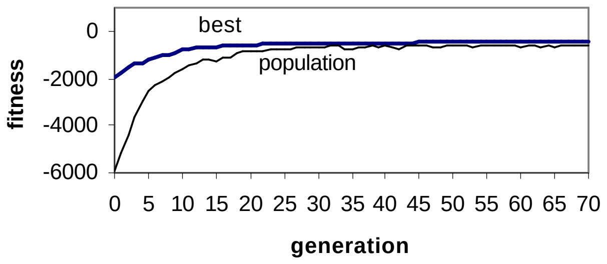
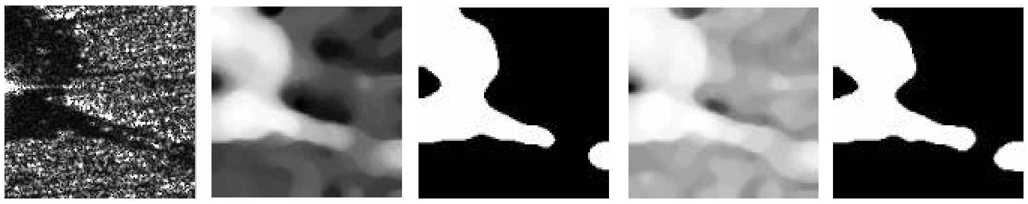


Figure 3.12. Fitness versus generation (river vs. field).



(a) river vs. field (b) feature image (normal GP) (c) ROI extracted (normal GP) (d) feature image (smart GP) (e) ROI extracted (smart GP)

Figure 3.13. Testing SAR image containing river.

The composite operator is applied to the testing image containing a river and field. Figure 3.13(b) shows the output of the composite operator from normal GP and Figure 3.13(c) shows the region extracted from Figure 3.13(a). The goodness of the region is 0.83. Figure 3.13(d) shows the output of the composite operator from smart GP and Figure 3.13(e) shows the region extracted. The goodness of the region is 0.86. There are some islands along with the river around them that are not extracted. The average running time of the best composite operators from normal GP on training and testing images is 33 seconds; the corresponding time of that from smart GP is 19 seconds.

3.5.4 Field Extraction

Two SAR images contain field and grass. Figure 3.14(a) and 3.14(b) show the original training image and the ground-truth. The training regions are from (17, 3) to (75, 61) and from (79, 62) to (124, 122). Extracting field from a SAR image containing field and grass is considered as the most difficult task among the five experiments, since the grass and field are similar to each other and some small regions between grasses are actually fields.

The generational GP was used to generate composite operators. For normal GP, the goodness value of the best composite operator in the initial population is 0.52 and the goodness value of the best composite operator in the final population is 0.78. Figure 3.14(c) shows the output image of the best composite operator on the whole training image and Figure 3.14(d) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.88. For smart GP, the fitness and goodness of the best composite operator in the initial population are –7936.2 and 0.39. The corresponding values in the finial population are –1999.4 and 0.79. Figure 3.14(e) shows the output image of the best composite operator on the whole training image and Figure 3.14(f) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.90.

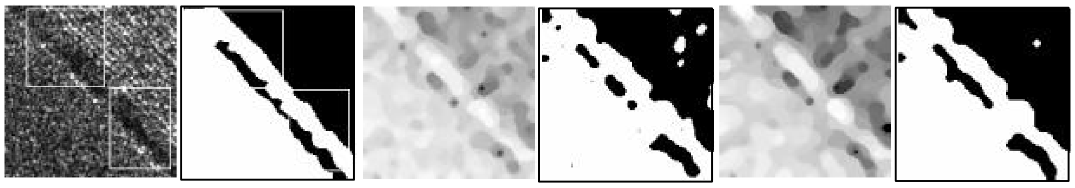

(a) field vs. grass (b) ground truth (c) feature image (normal GP) (d) ROI extracted (normal GP) (e) feature image (smart GP) (f) ROI extracted (smart GP)

Figure 3.14. Training SAR image containing field.

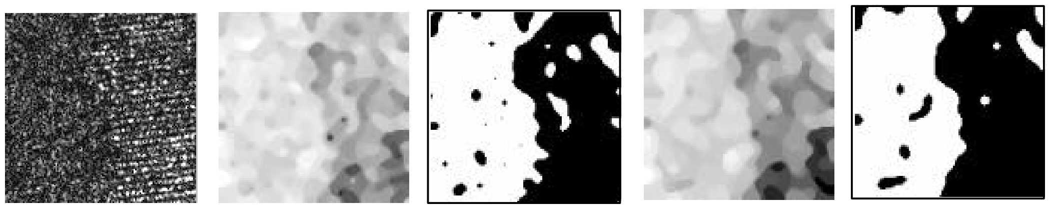

(a) field vs. grass (b) feature image (normal GP) (c) ROI extracted (normal GP) (d) feature image (smart GP) (e) ROI extracted (smart GP)

Figure 3.15. Testing SAR image containing field.

The composite operator is applied to the testing image containing field and grass shown in Figure 3.15(a). Figure 3.15(b) shows the output of the composite operator from normal GP and Figure 3.15(c) shows the extracted region with goodness value 0.81. Figure 3.15(d) shows the output of the composite operator from smart GP and Figure 3.15(e) shows the extracted region with goodness value 0.84. The average running time of the best composite operators from normal GP on training and testing images is 8 seconds; the corresponding time of that from smart GP is 12 seconds. The sizes of the best composite operators from normal GP and smart GP are 9 and 15, respectively.

3.5.5 Tank Extraction

In this subsection, GP is applied to synthesize features for the detection of military targets T72 tanks. Their SAR images are taken under different depression and azimuth angles and the size of the images is 80×80. The training image contains T72 tank under depression angle 17° and azimuth angle 135°, which is shown in Figure 3.16(a). The training region is from (19, 17) to (68, 66). The testing SAR image contains a T72 tank under depression angle 20° and azimuth angle 225°, which is shown in Figure 3.19(a). The ground-truth is shown in Figure 3.16(b).

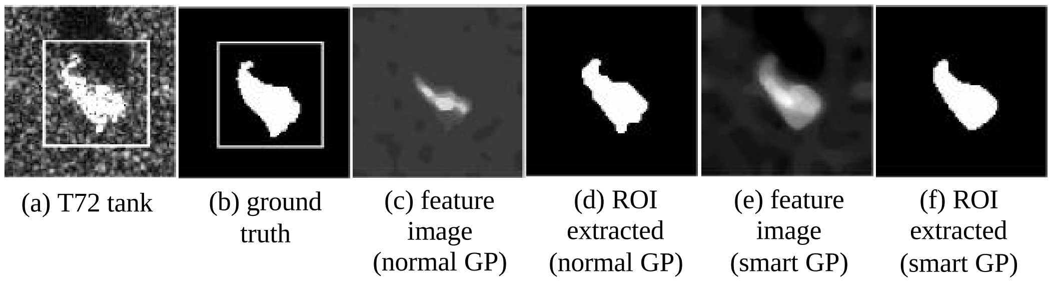

(a) T72 tank
(b) ground truth
(c) feature image (normal GP)
(d) ROI extracted (normal GP)
(e) feature image (smart GP)
(f) ROI extracted (smart GP)

Figure 3.16. Training SAR image containing tank.

The generational GP is applied to synthesize composite operators for tank detection. For normal GP, the goodness value of the best composite operator in the initial population is 0.65 and the goodness value of the best composite operator in the final population is 0.88. Figure 3.16(c) shows the output image of the best composite operator on the whole training image and Figure 3.16(d) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.88. For smart GP, the fitness and goodness of the best composite operator in the initial population are –807.2 and 0.54. The corresponding values in the finial population are –190.8 and 0.89. Figure 3.16(e) shows the output image of the best composite operator on the whole training image and Figure 3.16(f) shows the binary image after segmentation. The goodness value of the extracted ROI is 0.89. The best composite operator has 5 nodes and its depth is 4. It has one leaf node containing 3×3 maximum image. Two internal nodes are MED operator, which is useful in eliminating speckle noises in SAR images. It is shown in

Figure 3.17. Compared to smart GP, the best composite operator from normal GP has 28 nodes and its depth is 17. Figure 3.18 shows how the average fitness of the best composite operators and the average fitness of the populations over all 10 runs change as GP proceeds.

(MED (MED (SUBC (DIVC PFIM7))))

Figure 3.17. Learned composite operator tree in LISP notation.

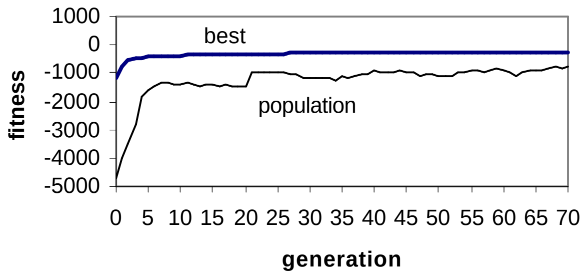


Figure 3.18. Fitness versus generation (T72 tank).

The composite operator is applied to the testing image containing T72 tank under depression angle 20° and azimuth angle 225°. Figure 3.19(b) shows the output of the composite operator from normal GP and Figure 3.19(c) shows the region corresponding to the tank. The goodness of the extracted ROI is 0.84. Figure 3.19(d) shows the output of the composite operator from smart GP and Figure 3.19(e) shows the region corresponding to the tank. The goodness of the extracted ROI is 0.84. The average running time of the best composite operators from normal GP on training and testing images is 3 seconds; the corresponding time of that from smart GP is 2 seconds. The results show that GP is very much capable of synthesizing composite operators for military target detection.

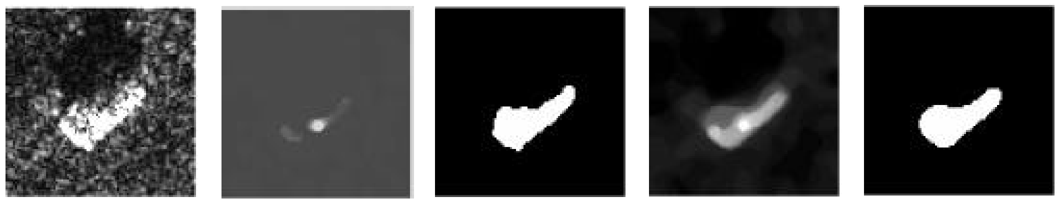
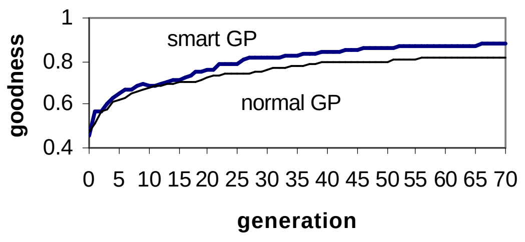

(a) T72 tank
(b) feature image (normal GP)
(c) ROI extracted (normal GP)
(d) feature image (smart GP)
(e) ROI extracted (smart GP)

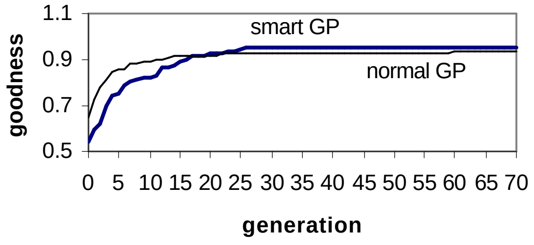
Figure 3.19. Testing SAR image containing tank.

3.5.6 Comparison between Normal GP and Smart GP

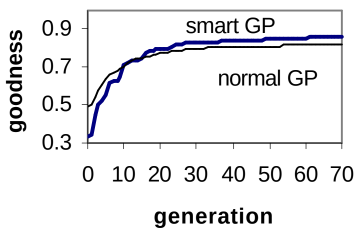
This subsection compares the performance of smart GP with that of normal GP. For objective comparison, only the average performance over all ten runs is used in comparison. Figure 3.20 shows how the average goodness of the best composites operators improves as normal GP and smart GP proceed. The thick line stands for the goodness of smart GP and the thin line stands for the goodness of normal GP. It shows that smart GP finds good composite operators more quickly. Table 3.1 shows the average goodness of the best composite operator in the initial and final populations.



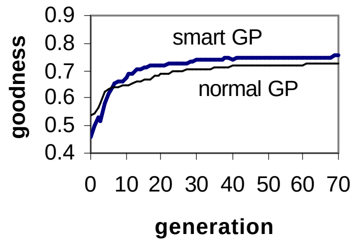
(a) road



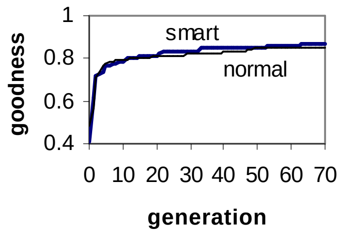
(b) lake



(c) river



(d) field



(e) tank

Figure 3.20. The average goodness of the best composite operators versus generation.

Table 3.1. The average goodness of the best composite operators from normal and smart GPs.

	Normal GP					Smart GP				
	Road	Lake	River	Field	Tank	Road	Lake	River	Field	Tank
Initial	0.47	0.64	0.49	0.54	0.49	0.46	0.54	0.34	0.46	0.41
Final	0.82	0.93	0.82	0.73	0.85	0.88	0.95	0.86	0.75	0.86

Table 3.2. The average size and performance of the best composite operators from normal and smart GPs.

	Normal GP					Smart GP				
	Road	Lake	River	Field	Tank	Road	Lake	River	Field	Tank
Size	29.4	28.4	27.6	20.2	24.6	24.6	11.8	16.8	14.9	5.7
Training	0.789	0.89	0.583	0.794	0.829	0.858	0.916	0.650	0.839	0.849
Testing	0.620 0.797	0.913	0.754	0.675	0.766	0.831 0.914	0.972	0.836	0.784	0.821

Table 3.2 shows the average size of the best composite operators from normal GP and smart GP. It also shows the average performances of the best composite operators on the whole training image and other testing image(s). It is obvious that the best composite operators learned by smart GP have better performance and smaller size. With smaller composite operators, the computational expense during testing is greatly reduced.

Table 3.3. Average running time of Normal GP and Smart GP.

	Road	Lake	River	Field	Tank
Normal GP	6915	2577	7951	3606	2686
Smart GP	10249	770	11035	5251	649

Table 3.3 shows the average running time of normal GP and smart GP. By intuition, the running time of smart GP should be much longer than that of normal GP, since in normal GP, only the output image of the root node is evaluated and smart GP evaluates the output image of each node of a composite operator. However, from Table 3.3, it can be seen that the difference between the running times is not as much as expected. In the experiments with lake and tank images, the running time of smart GP is much shorter. The reason lies in the code bloat problem of GP. In normal GP, a maximum size of composite operators (in this chapter, it is 30) is specified. At the later stage of the GP search, most of the composite operators have size equal or close to the maximum size allowed. In smart GP, the MDL-based fitness function takes the size of composite operators into the fitness evaluation process to avoid specifying a hard size limit. The difference between the sizes of composite operators is large, even at the later stage of the GP search. Although a few composite operators have size larger than the maximum size allowed in normal GP, many of them have size smaller than the specified size limit. If the maximum size allowed in normal GP is large, it can be expected that the running time of the normal GP will be longer than that of smart GP. Also, in the above experiments, the goodness threshold value is set 1.0 to force GP to finish the pre-specified number of generations. If goodness threshold value is smaller than 1.0, smart GP may run fewer generations, since it finds effective composite operators more quickly, thus reducing its running time.

3.6 Summary

In this chapter, smart crossover and smart mutation are proposed to improve the efficiency of genetic programming by identifying the effective components of composite operators and preventing them from being disrupted. The effective components are stored in a public library for later reuse. To address the well-known code bloat problem of GP, a new fitness function based on the minimum description length is designed to take the size of a composite operator into the fitness evaluation process. The new fitness function prevents composite operators from growing too large while at the same time imposing relatively less severe restrictions on the GP search. With MDL-based fitness function and the smart operators, GP can learn good composite operators more quickly, improving the efficiency of GP. Compared to normal GP, the composite operators learned by smart GP have better performance on the training and testing images and have smaller size, reducing the computational expenses during testing.

Chapter 4

GA-based Feature Selection for Object Detection

Automatic detection of potential objects in images is an important problem [3, 19]. A CFAR (constant false alarm rate) detector is commonly used to "prescreen" a synthetic aperture radar (SAR) image to localize possible object pixels [19]. Generally, object pixels correspond to bright spots caused by strong radar return from natural or man-made objects in SAR images. Parts of an image that are not selected are rejected from further computation. In the next stage of processing, regions of interest are further examined to distinguish man-made objects from natural clutter. Finally, a classifier such as a Bayesian classifier, a template matcher or a model-based recognizer is used to reject man-made clutter.

In chapters 2 and 3, genetic programming is applied to synthesize composite features from primitive features for object detection. The primitive features used there are domain-independent and not specific to a kind of imagery. The focus of this chapter is to select a minimal set of features from 20 available features to distinguish objects from natural clutter. 10 of these 20 features are simple and domain-independent features and the other 10 are specifically designed to process SAR images. The approach is based on a closed loop system involving GA based feature selection and a Bayesian classifier. GA uses a MDL-based fitness function that combines the number of features to be used and the error rate of the classifier. The results are presented using real SAR images. The experimental results show that the MDL-based fitness function is the most effective in selecting a minimal set of features to describe the data accurately compared to other three fitness functions, and the subset of features selected can greatly reduce the computational cost while at the same time maintaining the desired detection accuracy.

4.1 Motivation

In general, the goal of feature selection is to find the subset of features that produces the best object detection and recognition performance and requires the least computational effort. Feature selection is important to object detection and recognition systems mainly for three reasons:

First, using more features can increase system complexity, yet it may not always lead to higher detection/recognition accuracy. Sometimes, many features are available to a detection/recognition system. However, these features are not independent and may be correlated. A bad feature may greatly degrade the performance of a system. Thus, selecting a subset of good features is important.

Second, features are selected by a learning algorithm during the training phase. The selected features are used as a model to describe the training data. Selecting many features means a complicated model being used to approximate the training data. According to minimum description length (MDL) principle, a simple model is better than a complex model [12]. Since the training data may be corrupted with a variety of noises, a complex model may overfit the training data and may be sensitive to noises, leading to bad performance on unseen test data. In this chapter, GA is used to select as few features as possible to describe the training data effectively.

Third, using fewer features can reduce the computational cost, which is important for real-time applications. Also it may lead to better classification accuracy due to the finite sample size effect.

4.2 Related Research

Genetic algorithms (GAs) are widely used in image processing, pattern recognition and computer vision [3, 20, 21]. They are used to evolve morphological probes that sample the multi-resolution images [22], to generate image filters for target detection [23], to select good parameters of partial shape matching for occluded object recognition [24], to perform pattern clustering and classification [25], etc. GAs are also used to automatically determine the relative importance of many different features and to select a good subset of features available to the system [26].

Bhanu and Lee [27] present a closed loop image segmentation system, which incorporates a genetic algorithm to adapt the segmentation process to changes in image characteristics caused by variable environmental conditions such as time of day, time of year, clouds, etc. The segmentation problem is formulated as an optimization problem and the genetic algorithm efficiently searches the hyperspace of segmentation parameter combinations to determine the parameter set which maximizes the segmentation quality criteria in terms of edge-border coincidence, boundary consistency, pixel classification, object overlap and object contrast. Their experimental results demonstrate that genetic algorithm can continuously adapt the segmentation process to normal environmental variations to provide robust performance when interacting with a dynamic environment. Emmanouilidis et al. [28] discuss the use of multi-criteria genetic algorithms for feature selection. With multi-criteria fitness functions, genetic algorithm tries to minimize the number of features selected while maintaining the high classification accuracy. The algorithm is shown to yield a diverse population of alternative feature subsets with various accuracy and complexity trade-off. It is applied to select features for performing classification with fuzzy models and is evaluated on real-world data sets such as cancer data set in which each data point has 9 input features and one output label (malignant or benign). Estevez et al. [29] propose a genetic algorithm for selecting features for neural network classifiers. Their algorithm is based on a niching method to find and maintain multiple optima. They also introduce a new mutation operator to speed up the convergence of the genetic algorithm. Rhee and Lee [30] present an unsupervised feature selection method using a fuzzy-genetic approach. The method minimizes a feature evaluation index, which incorporates a weighted distance between a pair of patterns used to rank the importance of the individual features. A pattern is represented by a set of features and the task of genetic algorithm is to determine the weighting coefficients of features in the calculation of weighted distance. Matsui et al. [31] use genetic algorithm to select the optimal combination of features to improve the performance of tissue classification neural networks and apply their method to problems of brain MRI segmentation to classify gray matter/white matter regions.

Quilan and Rivest [32] explore the use of minimum description length principle for the construction of decision trees. The MDL defines the best decision tree to be the one that yields the minimum combined length of the decision tree itself plus the description of the misclassified data items. Their experimental results show that the MDL provides a unified framework for both growing and pruning the decision tree, and these trees seem to compare favorably with those created by other techniques such as C4 algorithm. Gao et al. [33] use MDL to determine the best model granularity such as the sampling interval between the adjacent sampled points along the curve of Chinese characters or the number of nodes in the hidden layer of a three layer feed-forward neural network. Their experiments show that in these two quite different settings the theoretical value determined using MDL coincides with the best value found experimentally. The key point of their work is that using MDL the optimal granularity of the model parameters can be computed automatically rather than being tuned manually.

In this chapter, genetic algorithm is used to select a good subset of features used for object detection in SAR images. The object detection task involves the selection of a subset of features to discriminate SAR images containing objects from those containing clutter. The method is a novel combination of genetic algorithm based optimization of a criterion function that involves classification error and the number of features that are used for the discrimination of object from natural clutter in SAR images. The criterion (fitness) function proposed in this chapter is based on minimum description length principle and it compares favorably with other three fitness functions. The joint distribution of features is assumed to follow Gaussian distribution. The criterion function is optimized in a closed-loop with a Bayesian classifier evaluating the performance of each set of features. The GA used in feature selection is adaptive in the sense that it can automatically adapt the parameters such as crossover rate and mutation rate based on the efficiency of GA search in the feature space. As compared to this work, the feature selection presented in [19, 34] for target vs. natural clutter discrimination measures exhaustively the performance of each combination of the features by the P_d (probability of detection) versus P_{fa} (probability of false alarm) plot produced by it. The higher the P_d and the lower the P_{fa}, the better the combination of features.

4.3 Technical Approach

The purpose of the genetic algorithm (GA) based feature selection approach presented in this chapter is to select a set of features to discriminate objects from natural clutter false alarms in SAR images. The approach includes four stages: rough object detection, feature extraction from the potential object regions, feature selection based on the training data and the final discrimination. The first stage is based on the Lincoln Lab ATR system and the second stage uses some features (first 10 of the 20 features) used in their system [19, 34, 35]. In the feature selection stage, GA is used to select a best feature subset, defined as a particular set of features, which is the best in discriminating the object from the natural clutter false alarm. The diagram for feature selection is given in Figure 4.1.

4.3.1 Feature Evaluation

Adding more features does not necessarily improve discrimination performance. An important goal is to choose the best set of features from the discriminating features that are available. Before we do the feature selection, it is appropriate to give a set of feature evaluation criteria, which measure the discrimination capability of each feature or a combination of several features.

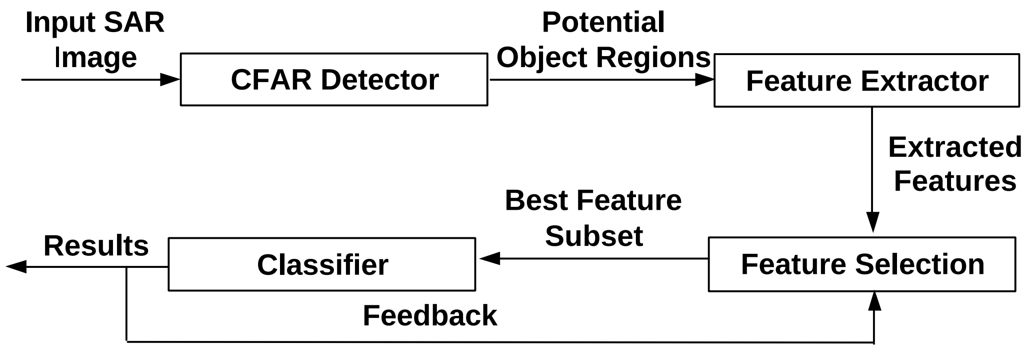


Figure 4.1. System diagram for feature selection.

- **Divergence:** Divergence is basically a form of the Kulback-Liebler distance measure between density functions. If we assume that the object as well as the natural clutter feature vectors follow the Gaussian distributions respectively, that is, $N(\mathbf{u}_1, \Sigma_1)$ and $N(\mathbf{u}_2, \Sigma_2)$, where $\mathbf{u}_1$ and $\mathbf{u}_2$ are mean values and Σ_1 and Σ_2 are covariance matrices, respectively, the divergence can be computed as follows:

$$d_{12} = \frac{1}{2}\, trace\left\{ \Sigma_1^{-1}\ \Sigma_2 + \Sigma_2^{-1}\ \Sigma_1 - 2\ I \right\} + \frac{1}{2} (\mathbf{u}_1 - \mathbf{u}_2)^T \left(\Sigma_1^{-1} + \Sigma_2^{-1} \right) (\mathbf{u}_1 - \mathbf{u}_2) \tag{4.1}$$

One major drawback of the divergence d_{12} is that it is not easily computed, unless the Gaussian assumption is employed. For SAR imagery, the Gaussian assumption itself is in question.

- **Scatter Matrices:** These criteria are based upon the information related to the way feature vector samples are scattered in the *l*-dimensional feature space. Two kinds of scatter matrices are defined. They are within-class scatter matrix and between-class scatter matrix. Within-class scatter matrix for *M* classes is, $S_w = \sum_{i=1}^{M} P_i S_i$, where S_i is the covariance matrix for class w_i and P_i is the a priori probability of class w_i. S_w matrix measures how feature vector samples are scattered within each class. Between-class scatter matrix S_b is defined as: $S_b = \sum_{i=1}^{M} P_i (\mathbf{u}_i - \mathbf{u}_0)(\mathbf{u}_i - \mathbf{u}_0)^T$, where $\mathbf{u}_0$ is the global mean vector and $\mathbf{u}_i$ is the mean for each class, $i = 1, \ldots, M$. The between-class scatter matrix measures how the feature vector samples are scattered between different classes. Based on the different combinations of these two scatter matrices, a set of class separability criteria can be derived; one such measure can be defined as: $J = \frac{|S_b|}{|S_w|}$. If the feature vector samples within each class are scattered compactly and the feature vector samples from different classes are far away from one another, we expect the value for *J* would be high. This also implies that the features we choose have high discrimination.

- **Feature vector evaluation using a classifier:** Another method for feature evaluation depends on a specific classifier. The task of feature selection is to select or determine a set of features, that when fed into a classifier, will let the classifier achieve the best performance. So it makes sense to relate the feature selection procedure with a particular classifier used. During the training time, the features extracted from the training data are available. What the feature selection algorithm does is to select a subset of these features and feed them into the classifier and see the classification result. Then the goodness of each feature subset is indicated by its classification error rate.

4.3.2 Various Criteria for Fitness Functions

GA is used to seek the smallest (or the least costly) subset of features for which the classifier's performance does not deteriorate below a certain specified level [26, 36]. The basic system framework is shown in Figure 4.1.

When the error of a classifier is used to measure the performance, a subset of features is defined as feasible if the classifier's error rate is below the so-called *feasibility threshold*. We search for the smallest subset of features among all feasible subsets. During the search, each subset can be coded as a *d*-element bit string (*d* is the total number of features). The *i*th element of the bit string assumes 0 if the *i*th feature is excluded from the subset and 1 if it is present in the subset.

In order for the GA to select a subset of features, a fitness function must be defined to evaluate the performance of each subset of features. GA explores the space of feature subsets to try to find a minimum subset of features with good classification performance.

4.3.2.1 Fitness Function Based on MDL

In this chapter, the classifier is fixed, which is a Bayesian classifier, but the set of features that is input into the classifier is a variable. In order to apply MDL to feature selection, the features selected by GA are viewed as the model used to describe the training data. Selecting more features means that a more complex model is used to approximate the data. Although a complex model may have perfect performance on the training data, it may not be a good model, since it may be overly sensitive to statistical irregularities and idiosyncrasies of the data and causes accidental noise to be modeled as well, leading to the poor performance on the unseen test data.

To fix the above problem, minimum description length principle is used to prevent the overfitting of the training data by an overly complex model. Roughly speaking, the MDL states that among all the models approximating the data to or above certain accuracy, the simplest one is the best one. To restrict the model from growing too complex while maintaining the description accuracy, the cost of describing a set of data with respect to a particular model is defined as the sum of the length of the model and the length of the data when encoded using the model as a predictor for the data. The description length of data-to-model error is defined as the combined length of all the data items failed to be described by the model. GA is used to select the subset of features minimizing the above cost. Here, both description lengths are measured in bits and the details of the coding techniques are relevant. The trade-off between simplicity and complexity of both lengths is that if a model is too simple, it may not capture the characteristics of the data and lead to increased error-coding length, if a model is too complicated, it may model the noise and become too sensitive to minor irregularities to give accurate prediction of unseen data. MDL states that among the given set of models, the one with the minimum combined description length of both the model and data-to-model error is the best model and can perform best on the unseen test data.

Based on MDL, the following fitness function is proposed for GA to maximize:

$$F(c_i) = -(k\log(f) + n_e\log(n)) \tag{4.2}$$

where c_i is a chromosome coding the selected set of features, f is the total number of features extracted from each training data, k is the number of features selected (c_i has k bits of 1 and $f - k$ bits of 0), n is the total number of data items in the training set and n_e is the number of data items misclassified. It is easy to see that the fewer the number of features selected and smaller the number of data items misclassified, the larger the value of fitness function.

We now give a brief explanation of the above fitness function. Suppose a sender and a receiver both know all the data items and their order in the training set and also they agree in advance on the feature extractor used to extract the f features from each data item and the classifier used to classify each data based on the features extracted. But only the sender knows the label (object or clutter) of each data item. Now, the sender wants to tell the receiver the label of each data item. One simple approach to do this is to send a bit sequence of n bits where 1 represents an object and 0 represents a clutter. If n is large, then the communication burden will be heavy. In order to reduce the number of bits to be transmitted, in an alternative approach, the sender can tell the receiver which features can be used to classify the data, since the receiver can extract the features and apply the classifier on the features extracted to get the label of each data item. There are a total of f features and *log(f)* bits are needed to encode the index of each feature. If k features are selected, *k log(f)* bits are needed in order to inform the receiver which features should be extracted. However, some data items may be misclassified, so the sender needs to tell the receiver which data items are misclassified so that the receiver can get the correct labels of all the data in the training set. Since there are a total of n data items, *log(n)* bits are needed to encode the index of each data item. If n_e data items are misclassified, then $n_e log(n)$ bits are needed to convey to the receiver the indices of these misclassified data items. If the set of features selected is effective in discriminating objects from clutters, n_e is very small, thus the number of bits needs to be transmitted is much smaller than n.

4.3.2.2 Other Fitness Functions

Three other fitness functions are also used to drive GA and their performances are compared with that of the fitness function based on MDL.

In order to define two other fitness functions, we first define the following penalty function [36]:

$$p(e) = \frac{\exp((e-t)/m) - 1}{\exp(1) - 1} \tag{4.3}$$

where *e* is the error rate (the number of misclassified data item divided by the total number of data items in the training set) of the classifier, *t* is the feasibility threshold and *m* is called the “tolerance margin”. In this chapter, t = 0.01 and m = 0.005. It can be seen that if e < t, p(e) is negative and as *e* approaches zero, *p*(*e*) slowly approaches its minimal value. Note also that *p*(*t*) = 0 and *p*(*t* + *m*) = 1. For greater values of the error rate, this penalty function rises quickly toward infinity.

The *second* fitness function is defined as follows:

$$F(c_i) = -p(e) \tag{4.4}$$

This fitness function considers only the error rate of the classifier and does not care about how many features are selected. It can be predicted that this fitness function may lead to the selection of many features.

The *third* fitness function takes the complexity of a model, that is the number of features selected, into consideration. It combines the complexity of the model and its performance on the training data and is defined as follow:

$$F(c_i) = -(g \times number\_of\_features + (1-g)p(e)) \tag{4.5}$$

where γ ranges from 0 to 1 and determines the relative importance of the number of features selected and the error rate of the classifier. If we want to use fewer features, we can assign a large value to γ; if we think lower error rate is more important, we can assign a small value to γ. In the following experiments, γ takes value 0.1, 0.3 and 0.5.

The *fourth* fitness function is defined as follows:

$$F(c_i) = -(g\frac{k}{20} + (1-g)e) \tag{4.6}$$

where *k* is the number of features selected by GA and γ ranges from 0 to 1 and is a parameter that determines the relative importance of the number of feature selected and the error rate of the classifier.

4.3.3 System Description

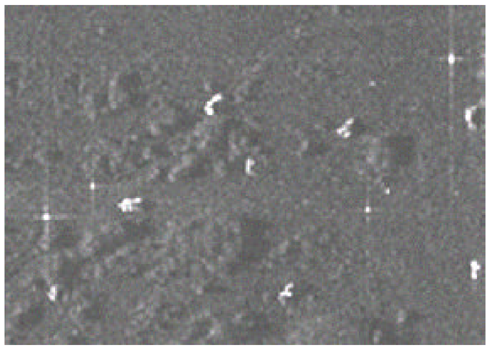
4.3.3.1 CFAR Detector

A two-parameter CFAR detector is used as a prescreener to identify potential objects in the image on the basis of radar amplitude. A guard area around a potential object pixel is used for the estimation of clutter statistics. The amplitude of the test pixel is compared with the mean and standard deviation of the clutter according to the following rule:

$$\frac{X_t - \hat{u}_c}{\hat{s}_c} > K_{CFAR} \Rightarrow \text{object} \quad , \text{ otherwise clutter} \tag{4.7}$$

where X_t is the amplitude of the test pixel, $\hat{u}_c$ is the estimated mean of the clutter amplitude, $\hat{s}_c$ is the estimated standard deviation of the clutter amplitude, and K_{CFAR} is a constant threshold value that defines the false-alarm rate.

Only those test pixels whose amplitude is much higher than that of the surrounding pixels are declared to be object pixels. The higher we set the threshold value of K_{CFAR}, the more a test pixel must stand out from its background for it to be declared as an object pixel. Because a single object can produce multiple CFAR detections, the detected pixels are grouped together if they are within an object-sized neighborhood. The CFAR detection threshold in the prescreener is set relatively low to obtain a high initial probability of object detection. It is the responsibility of the discriminator to capture and reject those escaping clutter false alarms from the prescreener stage. An example SAR image and the corresponding detection results are shown in Figure 4.2.


(a) Example SAR image.

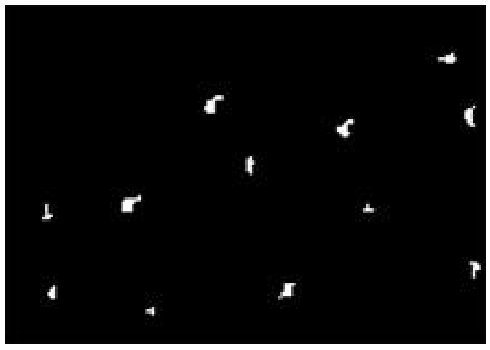

(b) Detection result.

Figure 4.2. SAR image and the CFAR detection result.

4.3.3.2 Feature Extractor

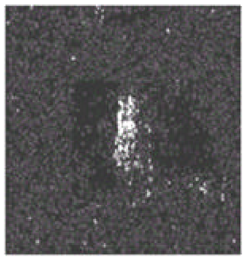
First, an object-sized rectangular template is used to determine the position and orientation of the detected object [37]. The algorithm slides and rotates the template until the energy within the template is maximized. Then a set of features is extracted from the object-sized template or the region of interest. This set of features is used to discriminate the objects from the natural clutters. First ten features are the same as those used in [19]. All the features from eleven to twenty are not used in their work, they are general features used in pattern recognition and object recognition.

- **The standard-deviation feature (feature 1):** The standard deviation of the data within the template is a statistical measurement of the fluctuation of the pixel intensities. If we use $P(r,a)$ to represent the radar intensity in power from range *r* and azimuth *a*, the standard deviation can be calculated as follows:

$$s = \sqrt{\frac{S_2 - \frac{S_1^2}{N}}{N-1}} \qquad \text{where} \qquad \begin{aligned} S_1 &= \sum_{r,\,a \in region} 10\log_{10} P(r,a) \\ S_2 &= \sum_{r,\,a \in region} [10\log_{10} P(r,a)]^2 \end{aligned} \tag{4.8}$$

and N is the number of pixels in the region.

Objects usually exhibit much larger standard deviation than natural clutters, as illustrated by Figure 4.3.


(a) A typical object image with standard deviation 5.2832.

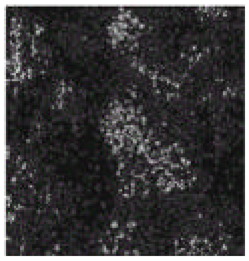

(b) A typical natural clutter image with standard deviation 4.5187.

Figure 4.3. Example of the standard deviation feature.

- **The fractal dimension feature (feature 2):** The fractal dimension of the pixels in the region of interest provides information about the spatial distribution of the brightest scatterers of the detected object. It complements the standard-deviation feature, which depends only on the intensities of the scatterers, not on their spatial locations.

The first step in applying the fractal-dimension concept to a radar image is to select an appropriately sized region of interest, and then convert the pixel values in the region of interest to binary. One method of performing this conversion is to select the *N* brightest pixels in the region of interest and convert their values to 1, while converting the rest of pixel values to 0. Based on these *N* brightest pixels, the fractal dimension is defined by using the following formula:

$$\dim = -\frac{\log M_1 - \log M_2}{\log 1 - \log 2} = \frac{\log M_1 - \log M_2}{\log 2} \tag{4.9}$$

where M_1 represents the minimum number of 1-pixel-by-1-pixel boxes that cover all *N* brightest pixels in the region of interest (This number is obviously equal to *N*) and M_2 represents the minimum number of 2-pixel-by-2-pixel boxes required to cover all *N* brightest pixels.

The bright pixels for a natural clutter tend to be widely separated, thus produce a low value for the fractal dimension, while the bright pixels for an object tend to be closely bunched, thus a high value for the fractal dimension is expected, which is illustrated by Figure 4.4. Figure 4.4(a) shows an object image. In Figure 4.4(b), the 50 brightest pixels from the object image are tightly clustered, and 22 2×2-pixel boxes are needed to cover them, which results in a high fractal dimension of 1.2. Figure 4.4(c) shows a natural clutter image. In Figure 4.4(d), the 50 brightest pixels from this natural clutter are relatively isolated, and 46 2×2-pixel boxes are needed to cover them, which results in a low fractal dimension of 0.29.

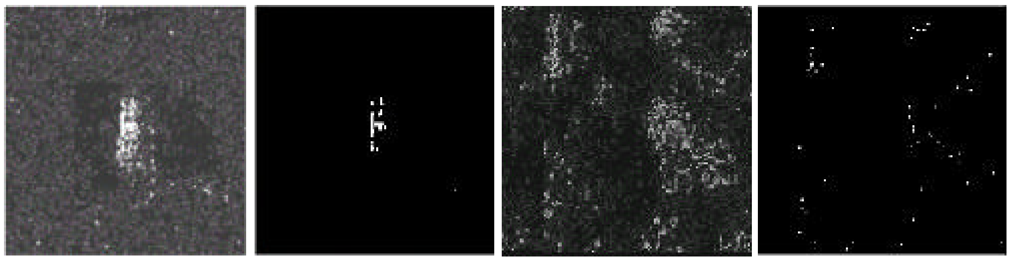


Figure 4.4. Example of the fractal dimension feature.

- **Weighted-rank fill ratio feature (feature 3):** This textual feature measures the percentage of the total energy contained in the brightest scatterers of a detected object. The weighted-rank fill ratio is defined as follows:

$$h = \frac{\sum_{k\ brightest\ pixels} P(r,a)}{\sum_{all\ pixels} P(r,a)} \tag{4.10}$$

This feature attempts to exploit the fact that power returns from most objects tend to be concentrated in a few bright scatters, whereas power returns form natural-clutter false alarms tend to be more diffuse. The weighted-rank fill ratio values of the object in Figure 4.3(a) and the clutter in Figure 4.3(b) are 0.3861 and 0.2321 respectively.

- **Size-related feature (features 4 - 6):** The three size-related features utilize only the binary image created by the morphological operations on the CFAR detection result.

1. The mass feature is computed by counting the number of pixels in the morphological blob.
2. The diameter is the length of the diagonal of the smallest rectangle that encloses the blob.
3. The square-normalized rotational inertia is the second mechanical moment of the blob around its center of mass, normalized by the inertia of an equal mass square.

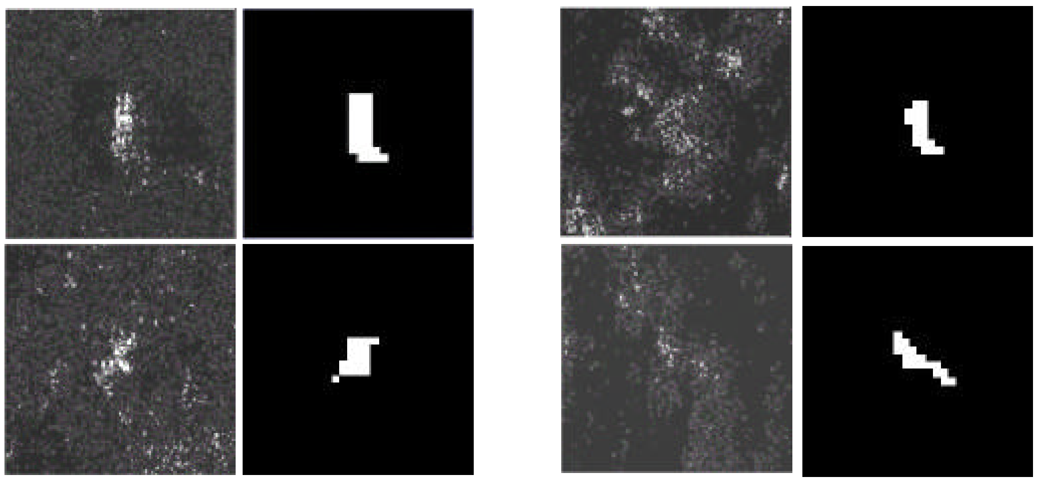
In the experiments, I find the size features are not effective in scenarios where the objects are partially occluded or hidden. After the prescreener stage, the size and the shape of the detected morphological blob can be arbitrary. For the clutter, there is also no ground to assert that the resulting morphological blob will exhibit a certain amount of coherence. The experimental results in Figure 4.5 show the arbitrariness of the morphological blobs for objects as well as clutters.

- **The contrast-based features (features 7 - 9):** The CFAR statistics is computed for each pixel in an object-shaped blob to create a CFAR image. Then the three features can be derived as follows:

1. The maximum CFAR feature is the maximum value of pixels in the CFAR image contained within an object-sized blob**.**

2. The mean CFAR feature is the average pixel value of pixels in the CFAR image taken over an object-shaped blob**.**

3. The percent bright CFAR feature is the percentage of pixels within an object-sized blob that exceed a certain CFAR value**.**



(a) The left-hand side figures represent the object images and the right-hand figures represent their corresponding morphological blobs.

(b) The left-hand side figures represent the clutter images and the right-hand figures represent their corresponding morphological blobs.

Figure 4.5. Examples of the size feature for (a) object and (b) clutter.

The maximum CFAR feature, the mean CFAR feature and the percent bright CFAR feature values of the object in Figure 4.3(a) are 55.69, 5.53 and 0.15, respectively, and these feature values of the clutter in Figure 4.3(b) are 10.32, 2.37 and 0.042, respectively. We can see that CFAR feature values for an object are much larger than those for a natural clutter false alarm.

- **The count feature (feature 10):** The count feature is very simple; it counts the number of pixels that exceeded the threshold *T* and normalize this value by the total possible number of pixels in an object blob. The threshold *T* is set to the quantity corresponding to the 98*th* percentile of the surrounding clutter. The count feature values of the object in Figure 4.3(a) and the clutter in Figure 4.3(b) are 0.6 and 0.1376, respectively. We can see that the count feature value for an object is much larger than that for a natural clutter false alarm. This makes sense because the intensity values of the pixels belonging to an object stand out from the surrounding clutter.

The following ten features (four projection features, three distance features and three moment features) are common features used in image processing and object recognition. They are extracted from binary images resulting from the CFAR detection. In these images, foreground pixels (pixels with value 1) are potential object pixels.

- **Projection features (features 11 – 14):** four projection features are extracted from each binary image:

1. *horizontal projection feature:* project the foreground pixels on a horizontal line (x axis of image) and compute the distance between the leftmost point and the rightmost point.
2. *vertical projection feature:* project the foreground pixels on a vertical line (y axis of image) and compute the distance between the uppermost point and the lowermost point.
3. *major diagonal projection feature:* project the foreground pixels on the major diagonal line and compute the distance between the upper leftmost point and the lower rightmost point.
4. *minor diagonal projection feature:* project the foreground pixels on the minor diagonal line and compute the distance between the lower leftmost point and the upper rightmost point.

The average values of horizontal, vertical, major and minor diagonal projection features of all the clutter images, I collected, are approximately 60.0, 60.0, 90.0 and 90.0, respectively. Their corresponding values for the object images are 34.5, 29.5, 46.7 and 47.8, respectively. It can be seen that the feature values for the clutters are larger than those for the objects. This result is reasonable, since the bright pixels of a natural clutter tend to be widely separated. This has already been shown by the fractal dimension feature value.

- **Distance features (features 15 – 17):** three distance features are extracted from each binary image. Before computing distance features, the centroid of all the foreground pixels in a binary image is computed first.

1. *minimum distance:* compute the distance from each foreground pixel to the centroid and select the minimum one.
2. *maximum distance:* compute the distance from each foreground pixel to the centroid and select the maximum one.

3. *average distance:* compute the distance from each foreground pixel to the centroid and get the average value of all these distances.

The average values of minimum, maximum and average distance features of all the clutter images I collected are approximately 40.0, 70.0 and 60.0, respectively. Their corresponding values of the object images are 3.8, 26.7 and 11.5, respectively. It can be seen that the feature values for the clutters are larger than those for the objects. This result is reasonable, since the bright pixels of a natural clutter tend to be widely separated.

- **Moment features (features 18 – 20):** three moment features are extracted from each binary image. All three moments are central moments, so before computing moment features, the centroid of all the foreground pixels in a binary image is computed first.

The central moments can be expressed as:

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\bar{x})^p(y-\bar{y})^q\,dxdy \qquad (4.11)$$

where $(\bar{x},\bar{y})$ is the centroid and p and q are integers.

Moments m_{20}, m_{02} *and* m_{22} are also call horizontal, vertical and diagonal second-order moment features, respectively. The average values of horizontal, vertical and diagonal second-order moment features of all the clutter images I collected are approximately 910.0, 910.0 and 374020.0, respectively. Their corresponding values of the object images are 80.5, 46.7 and 4021.6, respectively. It can be seen that the feature values for the clutters are larger than those for the objects. This result is reasonable, since the bright pixels of a natural clutter tend to be widely separated.

4.3.3.3 GAs for Feature Selection

The genetic algorithm is an optimization procedure that operates in binary search spaces (the search space consists of binary strings). A point in the search space is represented by a finite sequence of 0's and 1's, called a *chromosome*. The algorithm manipulates a finite set of chromosomes, the *population,* in a manner resembling the mechanism of natural evolution. Each chromosome is evaluated to determine its "fitness," which determines how likely the chromosome is to survive and breed into the next generation. The probability of survival is proportional to the chromosome's fitness value. Those chromosomes, which have higher fitness values are given more chances to "reproduce" by the processes of *crossover* and *mutation*. The function of crossover is to mate two parental chromosomes to produce a pair of offspring chromosomes. In particular, if a chromosome is represented by a binary string, crossover can be implemented by randomly choosing a point, called the crossover point, at which two chromosomes exchange their parts to create two new chromosomes. Mutation randomly perturbs the bits of a single parent to create a child. This procedure can increase the diversity of the population. Mutations can be performed by flipping randomly one or more bits in chromosomes. In this chapter, an adaptive genetic algorithm is implemented to automatically adapt the parameters such as crossover rate and mutation rate based on the performance of GA. To be specific, if the fitness value of the best individual is not improved for 3 or 5 generations in a row, GA will automatically raise the mutation rate to increase the diversity of the population. Also, elitism mechanism is adopted such that the best individual (set of features selected) is copied from generation to generation when performing reproduction.

In this chapter, there are 20 features as described earlier. Each feature is represented as a bit in the genetic algorithm. There are 2^{20} possible combinations of these features.

4.4 Experiments

SAR images from MSTAR public data (object and clutter data) are used to generate 1008 object chips (small SAR images containing object) and 1008 clutter chips (small SAR images containing clutter) of size 120×120. SAR images that are downloaded from the website of MIT Lincoln Lab are also used. From these SAR images, 40 object chips and 40 clutter chips of size 120×120 are generated. By adding these two sets of images, 1048 object chips and 1048 clutter chips are obtained. Some of the chips are used in training and the rest are used in testing. The chips used in training are randomly selected. The GA selects a good subset of features from the 20 features described previously to classify a SAR image chip into either an object or a clutter. The CFAR detector is used in the prescreener stage to detect the potential object regions. Since the ground truth about the identity of each chip (whether it is an object chip or a clutter chip) is known, this allows us to construct a set of training data (training object data and training natural clutter false alarm data) for feature selection. A set of 20 features is extracted from each potential object region and the feature selection is performed on the extracted features. Finally in the testing stage the selected features are used to discriminate the objects from the natural clutter false alarms in the testing object and clutter chips.

For the GA-based feature selection framework in this report, a Bayesian Classifier is adopted to classify the training data and the resulting error rate is used as the feedback into the feature selection algorithm. The size of the population is 100, the initial crossover rate is 0.8 and the initial mutation rate is 0.01. If the fitness value of the best individual is not improved for 3 generations in a row, GA increases the mutation rate by 0.02. The mutation rate is reset to 0.01 when the fitness of the best individual is increased. In order to reduce the training time, an error rate threshold ε is set. The GA stops when either the error rate of the best set of features selected is below the specified threshold ε or the mutation rate is increased above 0.09.

A series of experiments are carried out to test the efficacy of GA in feature selection. First, the MDL-based fitness function is used. Then the other three fitness functions are used. Finally, the performances of these fitness functions are compared and analyzed. In order to have an objective comparison of various experiments, the GA is invoked ten times for each experiment with the same set of parameters and the same set of training chips. Only the average performances are used for comparison.

4.4.1 MDL-based Fitness Function

Four experiments are performed with this fitness function. In the first experiment, 300 object chips and 300 clutter chips are used in training and 748 object chips and 748 clutter chips are used in testing, the error rate threshold value ε is 0.002; in the second experiment, 500 object chips and 500 clutter chips are used in training and 548 object chips and 548 clutter chips are used in testing, the error rate threshold value ε is 0.0015; in the third and fourth experiments, 700 object chips and 700 clutter chips are used in training and 348 object chips and 348 clutter chips are used in testing, the error rate threshold value ε is 0.0015 and 0.0011, respectively. The features selected during training are used for classification during testing. It is worth noting that the training chip set in the third and fourth experiments is the superset of that in the second experiment and the training chip set in the second experiment is the superset of that in the first experiment. The object and clutter chips used during training are selected at random.

Table 4.1 shows the experimental results where 300 object and 300 clutter chips are used in training. GA is invoked 10 times and each row records the experimental results from the corresponding invocation. The last row records the average results of 10 runs. The column “Best generation” records the generation number in which the best set of features is found and the column “Total generation” shows the total number of generations GA runs. It can be seen that although the training error rate is 0.003 in each run, different features are selected. From the testing results, we can observe that sometimes clutter chips are misclassified as object chips. In some runs, the same number of testing clutter chips are misclassified, but the clutter chips that are misclassified in each run are different. The testing results show that GA finds an effective set of features to discriminate object from clutter. Table 4.2 and Table 4.3 show the experimental results when 500 object and clutter chips and 700 object and clutter chips are used in training, respectively. The results in Table 4.2 are very good. On the average, 5.1 features are selected and both the training and testing error rate are very low. However, the results in Table 4.3 are not good. Although the training and testing error rates are low, 9.2 features are selected on the average. From Table 4.3, we can see that GA runs 4.9 generations on the average. It is clear that GA stops prematurely. The reason for the premature termination is that the error rate threshold value 0.0015 is high in this case, since there are 700 object chips and

Table 4.1. Experimental results with 300 training object and clutter chips. (MDL, equation (4.2); ε = 0.002)

Run No.	Best Genera-tion	Total Genera-tion	Num-ber of Fea-tures	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	29	47	4	0100101001 0000000000	0.003	1	1	0.001	0	2
2	9	27	6	0110001011 0000100000	0.003	1	1	0.011	0	16
3	10	28	4	0100001001 0100000000	0.003	1	1	0.011	0	16
4	43	61	4	0000001001 0100100000	0.003	1	1	0.005	0	7
5	19	37	4	0101001001 0000000000	0.003	1	1	0.017	0	25
6	13	31	4	0100001001 1000000000	0.003	1	1	0.007	0	10
7	23	41	4	0100001001 0010000000	0.003	1	1	0.011	0	16
8	6	24	6	0010011011 0000100000	0.003	1	1	0.011	0	16
9	17	35	5	0100001001 0011000000	0.003	1	1	0.003	0	5
10	11	29	5	0100001001 0010100000	0.003	1	1	0.005	0	7
ave	18	36	4.6		0.003	1	1	0.0082	0	12

Table 4.2. Experimental results with 500 training object and clutter chips. (MDL, equation (4.2); ε = 0.0015)

Run No.	Best generation	Total generation	Number of features	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						Object	Clutter		Object	Clutter
1	17	35	5	0100001001 1000100000	0.002	1	1	0.006	0	7
2	13	31	5	0100001001 0000001001	0.002	1	1	0.006	0	7
3	19	38	5	0100001001 0000011000	0.002	1	1	0.006	0	7
4	20	38	5	0100001001 0000011000	0.002	1	1	0.006	0	7
5	10	28	5	0100001001 0010100000	0.002	1	1	0.006	0	7
6	26	44	5	0100001001 1100000000	0.002	1	1	0.003	0	3
7	25	43	5	0100001001 0000010100	0.002	1	1	0.007	0	8
8	9	27	6	0000001011 0000011010	0.002	1	1	0.007	0	8
9	8	26	5	0100001001 0000011000	0.002	1	1	0.006	0	7
10	17	35	5	0001001001 0011000000	0.002	1	1	0.004	0	4
ave	16.4	34.5	5.1		0.002	1	1	0.0057	0	6.5

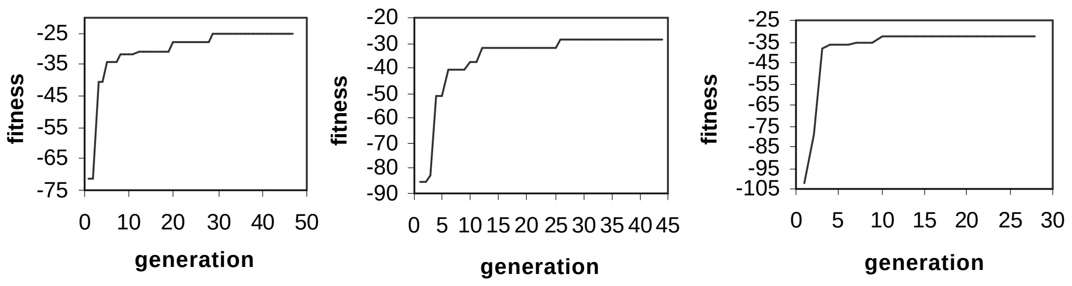
Table 4.3. Experimental results with 700 training object and clutter chips. (MDL, equation (4.2); ε = 0.0015)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	8	8	9	0101101001 1010001001	0.0014	1	1	0.006	0	4
2	9	9	10	1101001001 1010101010	0.0014	1	1	0.001	0	1
3	7	7	7	0000001011 0100101010	0.0014	1	1	0.012	0	8
4	2	2	10	1101001001 0110011010	0.0014	1	1	0.001	0	1
5	5	5	8	0100001001 0011111000	0.0014	1	1	0.007	0	5
6	2	2	7	1000011011 0100001000	0.0014	1	1	0.012	0	8
7	5	5	10	1101001001 0110101100	0.0014	1	1	0.001	0	1
8	3	3	10	1100101011 0101010001	0.0014	1	1	0.003	0	2
9	4	4	11	1101011001 1010111000	0.0014	1	1	0.001	0	1
10	4	4	10	1101001001 0011111000	0.0014	1	1	0.001	0	1
ave	4.9	4.9	9.2		0.0014	1	1	0.0045	0	3.2

Table 4.4. Experimental results with 700 training object and clutter chips. (MDL, equation (4.2); ε = 0.0011)

Run No.	Best Genera-tion	Total Genera-tion	Num-ber of Fea-tures	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-er
1	10	28	6	0001001001 1000001010	0.0014	1	1	0.004	0	3
2	19	37	5	0100001001 0000001010	0.0014	1	1	0.012	0	8
3	17	35	5	0100001001 0010100000	0.0014	1	1	0.01	0	7
4	16	34	6	0001011001 0010001000	0.0014	1	1	0.006	0	4
5	16	34	5	0100001001 0000011000	0.0014	1	1	0.01	0	7
6	19	37	5	0100001001 0010100000	0.0014	1	1	0.01	0	7
7	10	28	5	0100001001 0000010100	0.0014	1	1	0.01	0	7
8	15	33	5	0100001001 0000011000	0.0014	1	1	0.01	0	7
9	10	28	6	0100011001 1000010000	0.0014	1	1	0.007	0	5
10	23	41	5	0100001001 0000001001	0.0014	1	1	0.01	0	7
ave	15.5	33.5	5.3		0.0014	1	1	0.0089	0	6.1

700 clutter chips. In order to force GA to explore the search space, the error rate threshold value is lowered to 0.0011 and the results are shown in Table 4.4. These results are much better than those in Table 4.3. Only 5.3 features are selected on the average, although the average testing error rate is almost doubled. Considering both the test error rate and the number of features selected, the first run in Tables 4.1 and 4.4, and the sixth run in Table 4.2 yield the best results. Figure 4.6 shows how fitness changes as GA searches the feature subset space during these runs; Figure 4.7 shows how training error rate changes and Figure 4.8 shows how the number of features selected changes.



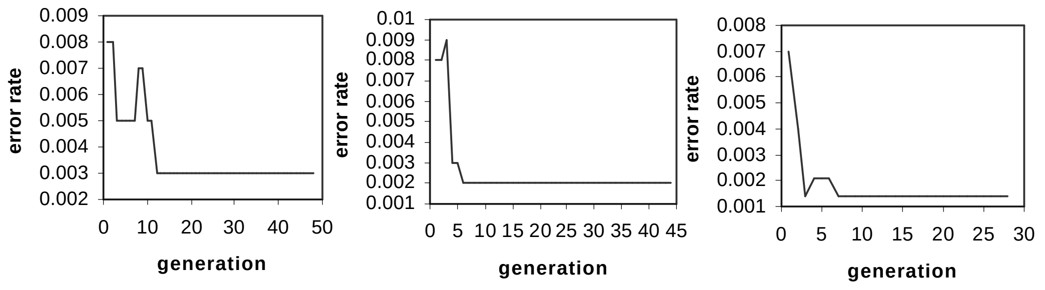
(a) 300 training object and clutter chips.

(b) 500 training object and clutter chips.

(c) 700 training object and clutter chips.

Figure 4.6. Fitness values vs. generation number.

From the above experiments, we can see that the MDL-based fitness function and adaptive GA are very efficient in feature selection. Only 4 to 6 features are selected on the average while the detection accuracy is kept high.

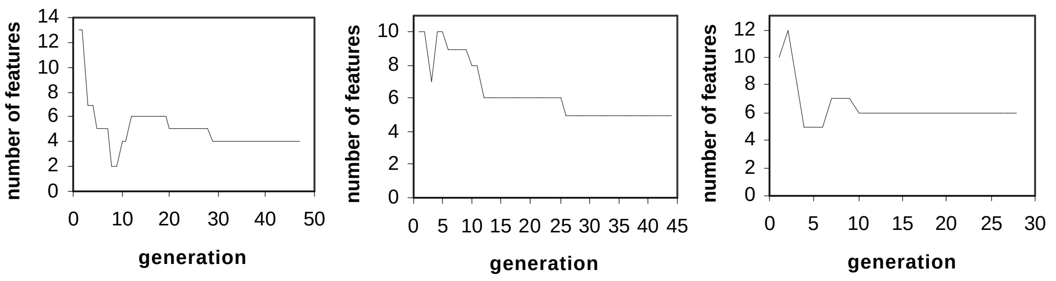


(a) 300 training object and clutter chips.

(b) 500 training object and clutter chips.

(c) 700 training object and clutter chips.

Figure 4.7. Error rates vs. generation number.



(a) 300 training object and clutter chips. (b) 500 training object and clutter chips. (c) 700 training object and clutter chips.

Figure 4.8. The number of features selected vs. generation number.

4.4.2 Other Fitness Functions

For the purpose of objective comparison, the training chips in the following experiments are the same as those in the second experiment above. 500 object chips and 500 clutter chips are used in training and 548 object chips and 548 clutter chips are used in testing.

First, function (4.4) is used as the fitness function and GA is invoked 10 times. The error rate threshold value is 0.0015. Table 4.5 shows the experimental results. This function is only dependent on the error rate, so GA found a set of features with very low error rate quickly. The selected features are shown by the "Number of features" and "Features selected" columns. However, since the number of features is not taken into consideration by the fitness function, many features are selected. More than 10 features are selected on the average over 10 runs.

Next, function (4.5) is used as the fitness function. Three experiments are performed with this function, and the values of γ are 0.1, 0.3 and 0.5 in these three experiments respectively. The error rate threshold is 0.0015. Since this function considers the number of features selected, you can imagine that few features will be selected. Tables 4.6, 4.7 and 4.8 show the corresponding experimental results when γ is 0.1, 0.3 and 0.5. From Table 4.6, we can see that since the training

Table 4.5. Experimental results with 500 training object and clutter chips. (penalty function, equation (4.4); ε = 0.0015)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Test-ing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	4	22	13	0111111011 1100111000	0.002	1	1	0.004	0	4
2	11	11	10	1011011011 0001100100	0.001	1	0	0.005	0	5
3	2	20	9	0101101001 1011000100	0.002	1	1	0.005	0	5
4	4	22	11	1010011011 0101011100	0.002	1	1	0.004	0	4
5	3	21	10	1110001011 1010010100	0.002	1	1	0.003	0	3
6	10	10	9	0011011011 0000110100	0.001	1	0	0.005	0	5
7	8	26	10	1101101001 0011010010	0.002	1	1	0.001	0	1
8	2	20	11	1110101011 0001001110	0.002	1	1	0.003	0	3
9	3	21	10	0110011011 1101100000	0.002	1	1	0.005	0	5
10	3	21	9	1110011011 0000110000	0.002	1	1	0.008	0	9
ave	5	19.4	10.2		0.0018	1	1	0.0043	0	4.4

Table 4.6. Experimental results with 500 training object and clutter chips. (penalty and # of features, equation (4.5); γ = 0.1; ε = 0.0015)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Test-ing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	18	36	2	0000001001 0000000000	0.005	2	3	0.024	0	26
2	12	30	2	0000001000 0000001000	0.007	1	6	0.007	0	8
3	17	35	2	0000001001 0000000000	0.005	2	3	0.024	0	26
4	20	38	2	0000001001 0000000000	0.005	2	3	0.024	0	26
5	16	34	2	0000001001 0000000000	0.005	2	3	0.024	0	26
6	11	29	2	0000001001 0000000000	0.005	2	3	0.024	0	26
7	15	33	2	0000001001 0000000000	0.005	2	3	0.024	0	26
8	17	35	2	0000001001 0000000000	0.005	2	3	0.024	0	26
9	14	32	2	0000001001 0000000000	0.005	2	3	0.024	0	26
10	12	30	2	0000001000 0000001000	0.007	1	6	0.007	0	8
ave	15.2	33.2	2		0.0054	1.8	3.6	0.0206	0	22.4

Table 4.7. Experimental results with 500 training object and clutter chips. (penalty and # of features, equation (4.5); $\gamma = 0.3$; $\varepsilon = 0.0015$)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	23	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	20	38	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	11	29	2	1000001000 0000000000	0.005	1	4	0.033	0	36
4	8	26	3	0000000010 0010010000	0.008	4	4	0.005	0	5
5	30	48	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	14	32	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	25	43	1	0000001000 0000000000	0.01	1	9	0.036	0	39
8	20	38	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	22	40	1	0000001000 0000000000	0.01	1	9	0.036	0	39
10	27	45	1	0000001000 0000000000	0.01	1	9	0.036	0	39
ave	20	38	1.3		0.0093	1.3	8	0.0326	0	35.3

Table 4.8. Experimental results with 500 training object and clutter chips. (penalty and # of features, equation (4.5); $\gamma = 0.5$; $\varepsilon = 0.0015$)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	17	35	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	29	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	22	40	1	0000001000 0000000000	0.01	1	9	0.036	0	39
4	15	33	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	32	50	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
8	23	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	9	27	2	0000000010 0000001000	0.012	5	7	0.011	0	12
10	23	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
ave	19.2	37.2	1.1		0.01	1.5	8.8	0.0335	0	36.3

Table 4.9. Experimental results with 500 training object and clutter chips. (error rate and # of features, equation (4.6); γ = 0.1; ε = 0.0015)

Run No.	Best Genera -tion	Total Genera-tion	Number of features	Features selected	Train -ing error rate	Number of errors		Testin g error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	21	39	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	16	34	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	14	32	2	0000100010 0000000000	0.01	7	3	0.006	0	7
4	25	43	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	13	31	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	17	35	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	17	35	2	0000100010 0000000000	0.01	7	3	0.006	0	7
8	33	51	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	22	40	1	0000001000 0000000000	0.01	1	9	0.036	0	39
10	12	30	1	0000001000 0000000000	0.01	1	9	0.036	0	39
ave	19	37	1.2		0.01	2.2	7.8	0.03	0	32.6

Table 4.10. Experimental results with 500 training object and clutter chips. (error rate and # of features, equation (4.6); γ = 0.3; ε = 0.0015)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	27	45	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	17	35	1	0000001000 0000000000	0.01	1	9	0.036	0	39
4	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	11	29	1	0000001000 0000000000	0.019	7	12	0.028	0	31
7	20	38	1	0000000010 0000000000	0.01	1	9	0.036	0	39
8	30	48	1	0000000010 0000000000	0.019	7	12	0.028	0	31
9	7	25	1	0000000010 0000000000	0.019	7	12	0.028	0	31
10	12	30	1	0000001000 0000000000	0.01	1	9	0.036	0	39
ave	15.7	33.7	1		0.013	2.8	9.9	0.0336	0	36.3

Table 4.11. Experimental results with 500 training object and clutter chips. (error rate and # of features, equation (4.6); $\gamma = 0.5$; $\varepsilon = 0.0015$)

Run No.	Best Genera-tion	Total Genera-tion	Number of features	Features selected	Train-ing error rate	Number of errors		Testing error rate	Number of errors	
						Ob-ject	Clut-ter		Ob-ject	Clut-ter
1	25	43	1	0000000010 0000000000	0.019	7	12	0.028	0	31
2	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	8	26	1	0000000010 0000000000	0.019	7	12	0.028	0	31
4	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	8	26	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	15	33	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	9	27	1	0000001000 0000000000	0.01	1	9	0.036	0	39
8	12	30	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	29	47	1	0000001000 0000000000	0.01	1	9	0.036	0	39
10	24	42	1	0000001000 0000000000	0.01	1	9	0.036	0	39
ave	15.2	33.2	1		0.013	2.2	9.4	0.0344	0	37.4

error rate is low, the number of features selected accounts for a large percentage of the value of the fitness function, forcing GA to select only 2 features in each run. However, the error rate for testing results is not encouraging. It is more than 0.02 on the average. When γ is 0.3, the number of features account for a larger part of the value of the fitness function than when γ is 0.1, forcing GA to select almost only one feature. Actually, in 8 runs, GA selects the best feature among all the 20 features (see Table 4.12) to discriminate object from clutter. When γ is 0.5, the number of features almost dominates the value of fitness function. The same phenomenon occurs and the experimental results are shown in Table 4.8.

Finally, function (4.6) is used as the fitness function. Three experiments are performed with this function, and the values of γ are 0.1, 0.3 and 0.5 in these three experiments, respectively. The error rate threshold is 0.0015. Like the function (4.5), this function considers both the number of features selected and the error rate. When γ is large, this function forces GA to select one feature. Usually, the best feature is selected (see Table 4.12). Tables 4.9, 4.10 and 4.11 show the corresponding experimental results when γ is 0.1, 0.3 and 0.5, respectively.

To show that GA selects the best feature when the number of features dominates the fitness function, the efficacy of each feature in discriminating objects from clutters is examined. The data used in examination are 500 object chips and 500 clutter chips used in the above training. The results are shown in Table 4.12.

From this table, it can be seen that the best feature (feature 7, the maximum CFAR feature) is selected by GA.

Table 4.12. Experimental results of using only one feature in discrimination. (object chips = 500, clutter chips = 500)

Feature	Error rate	Number of errors		Feature	Error rate	Number of errors	
		Object	Clutter			Object	Clutter
1	0.119	17	102	11	0.118	18	100
2	0.099	16	83	12	0.111	6	105
3	0.056	7	49	13	0.126	9	117
4	0.057	17	40	14	0.131	7	124
5	0.068	13	55	15	0.09	5	85
6	0.354	0	354	16	0.069	3	66
7	0.01	1	9	17	0.075	3	72
8	0.5	480	20	18	0.209	0	209
9	0.019	7	12	19	0.2	2	198
10	0.073	15	58	20	0.244	0	244

4.4.3 Comparison and Analysis

Figure 4.9 shows the average performances of the above experiments pictorially. The X-axis is the average number of features selected and the Y-axis is the average training error rate. The average number of features selected and average training error rate form a performance point and the performance is evaluated according to the location of a performance point. A good performance point should have lower values of both the number of features and the training error. The three points shown as circles are the performance points when the MDL-based fitness function is used and the rest are the performance points corresponding to other fitness functions.

From the above experimental results, we can see that GA is capable of selecting a good set of features to discriminate objects from clutters. The MDL-based fitness function is the best fitness function compared to three other functions. Fitness function (4.4) doesn't include the number of features. Although GA can find a good set of features quickly driven by this function, many features are selected. This greatly increases the computational complexity in the testing phase. Fitness functions (4.5) and (4.6) take the number of features selected into consideration. However, the number of features dominates the fitness function value, forcing GA to select only one or two features, leading to the unsatisfactory training and testing error rates. In order to balance the number of features selected and the error rate, parameter γ must be finely tuned. This is not an easy task and it usually takes a lot of time. The MDL-based fitness function is based on a sound theory and it balances these two terms very well. Only a few features are selected while the training and testing error rates are kept low.

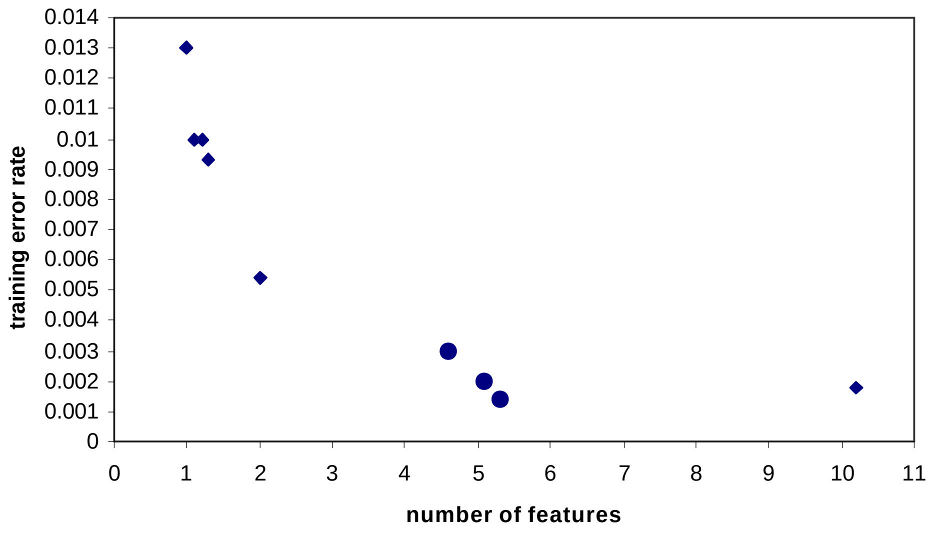


Figure 4.9. Average performances of various fitness functions.

In order to evaluate which features are more important than others using MDL-based approach, let us combine the results of the first, second and fourth experiments. Note that in the first, second and fourth experiments (shown in Tables 4.1, 4.2 and 4.4), GA are invoked for a total of 30 times. Table 4.13 shows the number of times each feature is selected in these 30 runs. It can be seen from Table 4.13 that the fractal dimension feature (feature 2), the maximum CFAR feature (feature 7) and the count feature (feature 10) are very useful in detecting objects in SAR images, and the standard deviation feature (feature 1) and the mean CFAR feature (feature 8) are not good. The major diagonal projection feature (feature 13), the minimum distance feature (feature 15), the maximum distance feature (feature 16) and the average distance feature (feature 17) have low utility while other features have very low utility. The results are consistent with those shown in Table 4.12. Considered individually, the maximum CFAR feature (feature 7) is the best feature (see Table 12) and selected (in combination with other features) in all the 30 runs.

Table 4.13. The number of times each feature is selected in Experiments 1, 2 and 4.

	Features																			
	1	**2***	3	4	5	6	**7***	8	9	**10***	11	12	13	14	15	16	17	18	19	20
Exp1	0	8	2	1	1	1	10	0	2	10	1	2	3	1	4	0	0	0	0	0
Exp2	0	8	0	1	0	0	10	0	1	10	2	1	2	1	2	4	5	1	1	1
Exp4	0	8	0	2	0	2	10	0	0	10	1	0	3	0	2	4	6	1	2	1
Total	0	24	2	4	1	3	30	0	3	30	4	3	8	2	8	8	11	2	3	2

4.5 Summary

In this chapter, a GA feature selection algorithm is applied to a specific application domain to discriminate objects from natural clutter false alarms in SAR images. Rough object detection, feature extraction, GA based feature selection and final discrimination are successfully implemented and good results are obtained. The experimental results show that GA selected a good subset of features. Also, an MDL-based fitness function is proposed and its performance is compared with three other fitness functions. The experimental results show that it balances the number of features selected and the error rate very well and it is the best fitness function compared to other three fitness functions.

Chapter 5

Learning Composite Features for Recognition Using Coevolutionary Genetic Programming

This chapter investigates synthesizing composite features for object recognition. The basic task of object recognition is to identify the kinds of the objects in an image, and sometimes the task may include estimating the pose of the recognized objects. One of the key approaches to object recognition is based on features extracted from images. These features capture the characteristics of the object to be recognized and are fed into a classifier to perform recognition. The quality of object recognition is heavily dependent on the effectiveness of features. However, it is difficult to extract good features from real images due to various factors, including noise. More importantly, there are many features that can be extracted. What are the appropriate features and how to synthesize composite features useful to the recognition from primitive features? The answers to these questions are largely dependent on the intuitive instinct, knowledge, experience and the bias of human experts.

In this chapter, the effectiveness of co-evolutionary genetic programming (CGP) in generating composite operator vectors for object recognition is investigated. The elements of a composite operator vector are synthesized composite operators. A composite operator is represented by a binary tree whose internal nodes are the pre-specified primitive operators and leaf nodes are primitive features. It is a way of combining primitive features. With each element evolved by a sub-population of CGP, a composite operator vector is cooperatively evolved by all the sub-populations. By applying composite operators, corresponding to each sub-population, to the primitive features extracted from images, composite feature vectors are obtained. These composite feature vectors are fed into a classifier for recognition. The primitive features are real numbers and they are designed by human experts based on the type of objects to be recognized. It is worth noting that the primitive operators and primitive features are decoupled from the CGP mechanism that generates composite operators. The users can tailor them to their own particular recognition task without affecting the other parts of the system.

Thus, the method and the recognition system are flexible and can be applied to a wide variety of images.

Motivation

The recognition accuracy of an automatic object recognition system is determined by the quality of the feature set used. Usually, it is the human experts who design the features to be used in recognition. Handcrafting a set of features requires human ingenuity and insight into the characteristics of the objects to be recognized and in general, it is very difficult to identify a set of features that characterize a complex set of objects. Typically, many types of features are explored before a recognition system can be built to perform the desired recognition task. There are a lot of features available and these features may be correlated, making the designing and selection of appropriate features a very time consuming and expensive process. Sometimes, it is even impossible to figure out and extract simple features that are effective in recognition directly from images. At this time, synthesizing composite features that are useful to the current recognition task from those simple primitive features available becomes extremely important. The process of synthesizing composite features can often be dissected into some primitive operations on the primitive features. It is usually the human experts who, replying on their knowledge, rich experience, figure out a smart way to combine these primitive operations to yield good composite features. The task of finding good composite features is equivalent to finding good points in the composite feature space. However, the ways of combining primitive features are almost infinite, leading to a huge composite feature space. It is obvious that a smart search strategy is a must in order to find good composite features in such a huge space. The human experts can try a very limited number of combination due to slow speed of human being and usually only the conventional combinations are tried due to knowledge, experience and even the bias of human experts. CGP, on the other hand, may try many unconventional combinations and in some cases it is these unconventional combinations that yield exceptionally good recognition performance. Also, the inherent parallelism of CGP and the speed of computers allow a much larger portion of the search space to be explored by CGP than that explored by human experts and this greatly enhances the chance of finding good composite features.

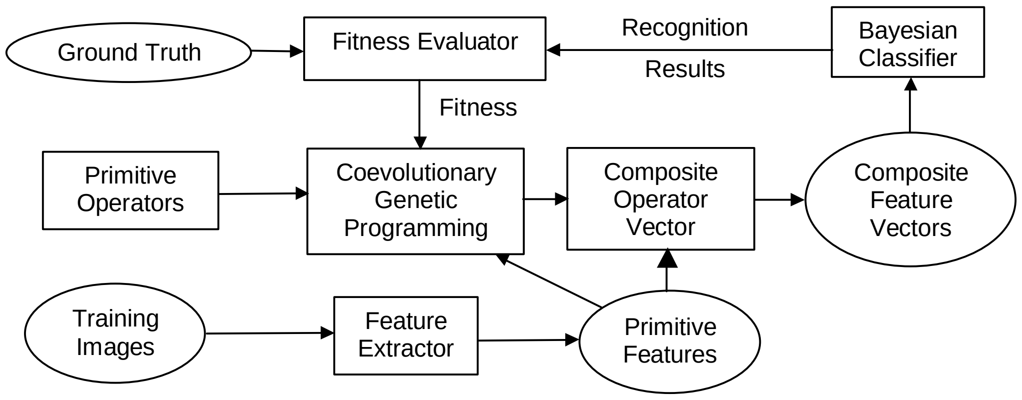
5.2 Related Research

Genetic programming (GP) has been used in image processing, object detection and recognition. Harris et al. [5] apply GP to the production of high performance edge detectors for 1D signals and image profiles. The method is also extended to the development of practical edge detectors for use in image processing and machine vision. Ebner et al. [38] use GP to automate the process of chaining a series of well-known image processing operators to perform image processing. Poli et al. [6] use GP to develop effective image filters to enhance and detect features of interest or to build pixel-classification-based segmentation algorithm. In chapters 2 and 3 and [39], GP is used to generate composite operators for object detection. The primitive operators and primitive feature images used are very basic and domain-independent, so the object detection method can be applied to a wide variety of images. The experimental results showed that GP is a viable way of synthesizing composite features from primitive features for object detection and ROI extraction. Howard et al. [9] applied GP to automatic detection of ships in low resolution SAR imagery using an approach that evolves detectors. The detectors are algebraic formulae involving the values at pixels belonging to a small region surrounding the pixel undergoing the test and the detectors evolved by GP compare favorably in accuracy to those obtained using a neural network. Roberts and Howard [10] use GP to develop automatic object detectors in infrared images. They present a multi-stage approach to address feature detection and object segregation and the detectors developed by GP do not require images to be pre-processed. Stanhope and Daida [8] used GP paradigms for the generation of rules for target/clutter classification and rules for the identification of objects. GP is used to select relevant features from a previously defined feature set and evolve logical expression on comparisons of the selected features to both real-valued constants and themselves to create a linear classifier. Krawiec and Bhanu [40, 41] present a method for the automatic synthesis of recognition procedures chaining elementary operations for computer vision and pattern recognition tasks based on cooperative coevolution and linear genetic programming. Each sub-population evolves a part of the recognition procedure and all the sub-populations coevolve the whole recognition procedure by selecting the best individual from each sub-population and chaining them together. Their experimental results show that genetic programming is effective in synthesizing recognition procedure from elementary image processing operations and they also show that coevolutionary genetic programming is superior to regular one-population genetic programming.

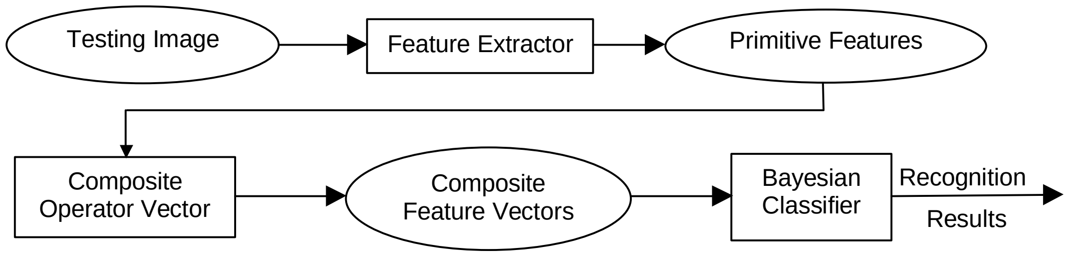
Unlike the work of Stanhope and Daida, the primitive operators in this chapter are not logical operators, but operators on real numbers and the composite operators are binary trees of primitive operators on real numbers, not binary trees of logical operators [70]. They use GP to evolve logical expressions and the final outcome of the logical expression determines the type of the object under consideration (for example, 1 means target and 0 means clutter). In this chapter, CGP is used to evolve composite feature vectors to be used by a Bayesian classifier [42] and each sub-population is responsible for evolving a specific composite feature in the composite feature vector. The classifier evolved by GP in their system can be viewed as a linear classifier, but the classifier evolved by CGP here is a Bayesian classifier determined by the composite feature vectors from training images. Unlike the work of Bhanu and Krawiec, composite operators in this chapter are binary tree of primitive operators and primitive features, whereas the recognition procedures in their system are linked list of simple image processing operations.

5.3 Technical Approach

In the CGP-based approach, individuals are composite operators represented by binary trees with primitive operators as internal nodes and primitive features as leaf nodes. The search space is the set of all possible composite operators. The search space is huge and it is extremely difficult to find good composite operators from this vast space unless one has a smart search strategy. The whole system is divided into training and testing parts, which are shown in Figure 5.1(a) and (b), respectively. During training, CGP runs on training images and evolves composite operators to obtain composite features. Since a Bayesian classifier is completely determined by the feature vectors from training images, so both the composite features and the classifier are learned by CGP.



(a) Training — Learning composite feature vectors and Bayesian classifier



(b) Testing — Applying learned composite feature vectors and Bayesian classifier to a test image

Figure 5.1. System diagram for object recognition using co-evolutionary genetic programming.

5.3.1 Design Considerations

- **The set of terminals:** The set of terminals are 20 primitive features used in chapter 4. The first 10 of them are designed by MIT Lincoln lab to capture the particular characteristics of synthetic aperture radar (SAR) imagery and are found useful for object detection. The other 10 features are common features used widely in image processing and computer vision. The 20 features are: (1) standard deviation of image; (2) fractal dimension and (3) weighted rank fill ratio of brightest scatterers; (4) blob mass; (5) blob diameter; (6) blob inertia; (7) maximum and (8) mean values of pixels within blob; (9) contrast brightness of blob; (10) count; (11) horizontal, (12) vertical, (13) major diagonal and (14) minor diagonal projections of scatterers; (15) maximum, (16) minimum and (17) mean distances of scatterers from their centroid; (18) moment μ_{20}, (19) moment μ_{02} and (20) moment μ_{22} of scatters. For detailed description, please refer to chapter 4.

- **The set of primitive operators:** A primitive operator takes one or two real numbers, performs a simple operation on them and outputs the result. Currently, 12 primitive operators shown in Table 5.1 are used, where a and b are real numbers and input to an operator and c is a constant real number stored in an operator.

Table 5.1. Twelve primitive operators.

Primitive Operator	Description	Primitive Operator	Description
ADD (a, b)	Add a and b.	ADDC (a, c)	Add constant value c to a.
SUB (a, b)	Subtract b from a.	SUBC (a, c)	Subtract constant value c from a.
MUL (a, b)	Multiply a and b.	MUL (a, c)	Multiply a with constant value c.
DIV (a, b)	Divide a by b.	DIVC (a, c)	Divide a by constant value c.
MAX2 (a, b)	Get the larger of a and b.	MIN2 (a, b)	Get the smaller of a and b.
SQRT (a)	Return $\sqrt{a}$ if a ≥ 0; otherwise, return $-\sqrt{-a}$.	LOG (a)	Return log(a) if a ≥ 0; otherwise, return – log(-a).

- **The fitness measure:** the fitness of a composite operator vector is computed in the following way: apply each composite operator of the composite operator vector on the primitive features of training images to obtain composite feature vectors of training images and feed them to a Bayesian classifier. The recognition rate of the classifier is the fitness of the composite operator vector. To evaluate a composite operator evolved in a sub-population (see Figure 5.2), the composite operator is combined with the current best composite operators in other sub-populations to form a complete composite operator vector where composite operator from the *ith* sub-population occupies the *ith* position in the vector and the fitness of the vector is defined as the fitness of the composite operator under evaluation. The fitness values of other composite operators in the vector are not affected. When sub-populations are initially generated, the composite operators in each sub-population are evaluated separately without being combined with composite operators from other sub-populations. After each generation, the composite operators in the first sub-population are evaluated first, then the composite operators in the second sub-population and so on.

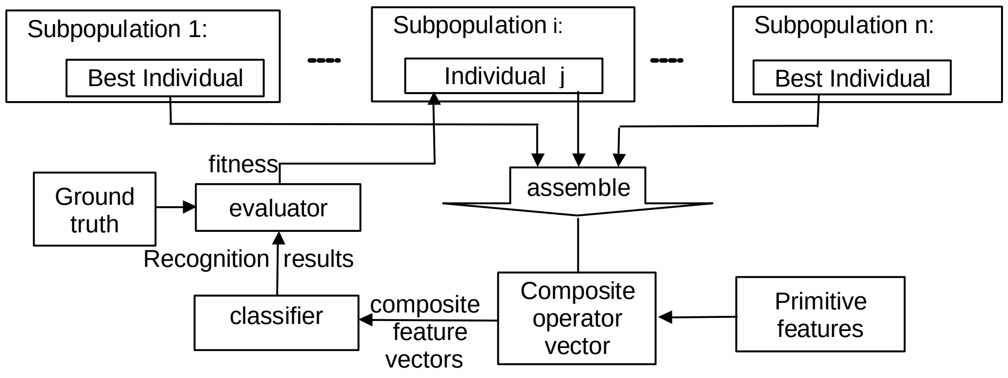


Figure 5.2. Computation of fitness of *jth* composite operator of *ith* subpopulation.

• **Parameters and termination**: The key parameters are the number of sub-population N, the population size M, the number of generations G, the crossover and mutation rates, and the fitness threshold. GP stops whenever it finishes the specified number of generations or the performance of the Bayesian classifier is above the fitness threshold. After termination, CGP selects the best composite operator of each sub-population to form the learned composite operator vector to be used in testing.

5.3.2 Selection, Crossover and Mutation

The CGP searches through the space of composite operator vectors to generate new composite operator vectors. The search is performed by selection, crossover and mutation operations. The initial sub-populations are randomly generated. Although sub-populations are cooperatively evolved (the fitness of a composite operator in a sub-population is not solely determined by itself, but affected by the composite operators from other sub-populations), selection is performed only on composite operators within a sub-population and crossover is not allowed between two composite operators from different sub-populations.

• **Selection**: The selection operation involves selecting composite operators from the current sub-population. In this chapter, tournament selection is used. The higher the fitness value, the more likely the composite operator is selected to survive.

• **Crossover**: Two composite operators, called parents, are selected on the basis of their fitness values. The higher the fitness value, the more likely the composite operator is selected for crossover. One internal node in each of these two parents is randomly selected, and the two subtrees rooted at these two nodes are exchanged between the parents to generate two new composite operators, called offspring. It is easy to see that the size of one offspring (i.e., the number of nodes in the binary tree representing the offspring) may be greater than both parents if crossover is implemented in such a simple way. To prevent code bloat, we specify a maximum size of a composite operator. If the size of one offspring exceeds the maximum size, the crossover is performed again until the sizes of both offspring are within the limit.

- **Mutation:** To avoid premature convergence, mutation is introduced to randomly change the structure of some composite operators to maintain the diversity of sub-populations. Candidates for mutation are randomly selected and the mutated composite operators replace the old ones in the sub-populations. There are three mutations invoked with equal probability:

1. Randomly select a node of the composite operator and replace the subtree rooted at this node by another randomly generated binary tree
2. Randomly select a node of the composite operator and replace the primitive operator stored in the node with another primitive operator randomly selected from the primitive operators of the same number of input as the replaced one.
3. Randomly selected two subtrees of the composite operator and swap them. Of course, neither of the two sub-trees can be a sub-tree of the other.

5.3.3 Generational Co-evolutionary Genetic Programming

Generational co-evolutionary genetic programming is used to evolve composite operators. The GP operations are applied in the order of crossover, mutation and selection. Firstly, two composite operators are selected on the basis of their fitness values for crossover. The two offspring from crossover are kept aside and won't participate in the following crossover operations on the current sub-population. The above process is repeated until the crossover rate is met. Then, mutation is applied to the composite operators in the current sub-population and the offspring from crossover. Finally, selection is applied to select some composite operators from the current sub-population and combine them with the offspring from crossover to get a new sub-population of the same size as the old one. In addition, an elitism replacement method is adopted to keep the best composite operator from generation to generation.

- **Generational Co-evolutionary Genetic Programming:**

```
0. randomly generate N sub-populations of size M and evaluate each composite
   operator in each sub-population individually.
1. for gen = 1 to generation_num do
2.     for i =1 to N do
3.         keep the best composite operator in sub-population $P_i$.
4.         perform crossover on the composite operators in $P_i$ until the crossover
           rate is satisfied and keep all the offspring from crossover.
```

```
5.          perform mutation on the composite operators in $P_i$ and the offspring
            from crossover with the probability of mutation rate.
6.          perform selection on $P_i$ to select some composite operators and combine
            them with the composite operators from crossover to get a new sub-
            population $P_i$' of the same size as $P_i$.
7.          evaluate each composite operator $C_j$ in $P_i$'. To evaluate $C_j$, select the cur-
            rent best composite operator in each of the other sub-populations, com-
            bine $C_j$ with those N-1 best composite operators to form a composite op-
            erator vecter where composite operator from the kth sub-population oc-
            cupy the kth position in the vector (k=1, …, N). Run the composite opera-
            tor vector on the primitive features of the training images to get compos-
            ite feature vectors and use them to build a Bayesian classifier. Feed the
            composite feature vectors into the Bayesian classifier and let the recogni-
            tion rate be the fitness of the composite operator vector and the fitness of
            $C_j$.
8.          let the best composite operator from $P_i$ replace the worst composite op-
            erator in $P_i$' and let $P_i$ = $P_i$'
9.          Form the composite operator vector consisting of the best composite op-
            erators from corresponding sub-populations and evaluate it. If its fitness
            is above the fitness threshold, goto 10.
        endfor // loop 2
    endfor // loop 1
10. select the best composite operator from each sub-population to form the
    learned composite operator vector and output it.
```

5.4 Experiments

Various experiments are performed to test the efficacy of genetic programming in generating composite features for object recognition. In this chapter, we show some selected examples. All the images used in the experiments are real synthetic aperture radar (SAR) images. These images are divided into training and testing images. 20 primitive features are extracted from each SAR image. CGP runs on primitive features from training images to generate a composite operator vector and a Bayesian classifier. The composite operator vector and the Bayesian classifier are tested against the testing images. It is worth noting that the ground truth is used only during training. The experiments are categorized into three classes: (1) distinguishing man-made military objects from natural clutters, (2) distinguishing between 3 kinds of man-made military objects and (3) distinguishing between 5 kinds of man-made military objects. For the purpose of objective comparison, CGP is invoked ten times for each experiment with the same set of parameters and the same set of training images. Only the average performances are used for comparison. Some of the parameters of CGP used throughout the experiments are shown in Table 5.2. The maximum size of a composite operator is 10 in experiment 1 and 20 in experiments 2 and 3. The constant real number c stored in some primitive operators ranges from –20 to 20.

Table 5.2. Parameters of CGP used throughout the experiments.

Sub-population size	50	Crossover rate	0.6
Number of generation	50	Mutation rate	0.05
Fitness threshold	1.0		

5.4.1 Distinguish Object from Clutter

- **Data:** The data used here are the same as those used in chapter 4. From MSTAR public real SAR images, 1048 SAR images containing objects and 1048 SAR images containing natural clutters are generated. These images have size 120×120 and are called object images and clutter images, respectively. An example object image and clutter image are shown in Figure 5.3, where white spots indicate scatterers with high magnitude. 300 object images and 300 clutter images are randomly selected as training images and the rest are used in testing.

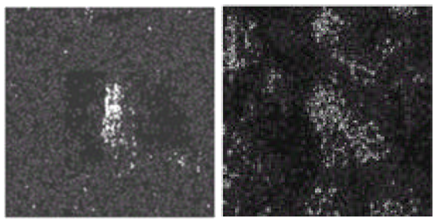

(a) An object image (b) A natural clutter image

Figure 5.3. Example object and clutter SAR images.

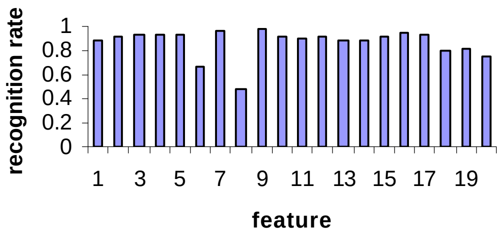


Figure 5.4. Recognition rates of 20 primitive features.

- **Experiment 1:** First, the efficacy of each primitive feature in discriminating the objects from the clutters is examined. Each kind of primitive features from training images is used to train a Bayesian classifier and the classifier is tested against the same kind of primitive features from the testing images. The results are shown in Figure 5.4 and Table 5.3. Feature contrast brightness of blob (9) is the best one with recognition rate 0.98.

To show the efficacy of CGP in synthesizing effective composite features, we consider three cases: only the worst two primitive features (blob inertia (6) and mean values of pixels within blob (8)) are used by CGP; five bad primitive features (blob inertia (6), mean values of pixels within blob (8), moments μ_{20} (18), μ_{02} (19) and μ_{22} (20) of scatters) are used by CGP; 10 common features (primitive features 11 to 20) not specifically designed to deal with SAR images are used by CGP during synthesis. The number of sub-populations is 3, which means the dimension of the composite feature vector is 3. The results are shown in Table 5.4 and Figure 5.5, where the horizontal coordinates are the number of primitive features used in synthesis and the vertical coordinates are the bins on the

Table 5.3. Recognition rates of 20 primitive features.

Feature Number	Primitive Feature	Recognition Rate	Feature Number	Primitive Feature	Recognition Rate
1	Standard deviation	0.88	11	Horizontal projection	0.90
2	Fractal dimension	0.91	12	Vertical projection	0.91
3	Weight-rank fill ratio	0.94	13	Major diagonal projection	0.89
4	Blob mass	0.94	14	Minor diagonal projection	0.88
5	Blob diameter	0.94	15	Minimum distance	0.92
6	Blob inertia	0.66	16	Maximum distance	0.95
7	Maximum CFAR	0.97	17	Mean distance	0.94
8	Mean CFAR	0.49	18	Moment μ_{20}	0.80
9	Percent bright CFAR	0.98	19	Moment μ_{02}	0.81
10	Count	0.92	20	Moment μ_{22}	0.75

left show the training results and those on the right show the testing results. The numbers above the bins are the average recognition rates over all ten runs. Then the number of sub-population is increased from 3 to 5. The same 2, 5 and 10 primitive features are used as building blocks by CGP to evolve composite features. The experimental results are shown in Table 5.5 and Figure 5.6.

Table 5.4. Experimental results with 3 sub-populations.

Runs	Recognition Rate					
	2f		5f		10f	
	Training	Testing	Training	Testing	Training	Testing
1	0.99	0.980	0.992	0.992	0.965	0.983
2	0.988	0.991	0.993	0.994	0.975	0.982
3	0.988	0.989	0.993	0.987	0.978	0.987
4	0.988	0.989	0.992	0.985	0.975	0.987
5	0.988	0.989	0.978	0.987	0.975	0.989
6	0.988	0.987	0.995	0.995	0.965	0.979
7	0.988	0.990	0.992	0.992	0.973	0.983
8	0.99	0.979	0.992	0.974	0.97	0.983
9	0.99	0.979	0.992	0.993	0.968	0.983
10	0.992	0.983	0.992	0.987	0.968	0.983
Average	0.989	0.986	0.991	0.989	0.971	0.984

From Figures 5.5 and 5.6, it is obvious that composite feature vectors synthesized by CGP are very effective. They are much better than the primitive features upon which they are built. Actually, if both features 6 and 8 from the training images jointly form 2-dimensional primitive feature vectors to train a Bayesian classifier for recognition, the recognition rate is 0.668 on the testing images; if features 6, 8, 18, 19, and 20 jointly form 5-dimensional primitive feature vectors, the recognition rate is 0.947; if all the last 10 primitive features are used, the recognition rate is 0.978. The average recognition rates of composite feature

Table 5.5. Experimental results with 5 sub-populations.

Runs	Recognition Rate					
	2f		5f		10f	
	Training	Testing	Training	Testing	Training	Testing
1	0.988	0.981	0.993	0.995	0.975	0.992
2	0.99	0.980	0.995	0.993	0.972	0.979
3	0.988	0.983	0.995	0.991	0.97	0.981
4	0.99	0.982	0.993	0.991	0.983	0.986
5	0.992	0.983	0.993	0.987	0.977	0.990
6	0.988	0.982	0.995	0.992	0.98	0.990
7	0.99	0.984	0.993	0.993	0.977	0.981
8	0.992	0.981	0.995	0.992	0.975	0.986
9	0.99	0.984	0.995	0.990	0.982	0.986
10	0.992	0.983	0.993	0.996	0.975	0.987
Average	0.99	0.982	0.994	0.992	0.977	0.986

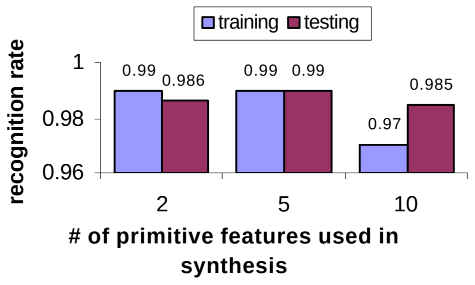


Figure 5.5. Experimental results with 3 sub-populations.

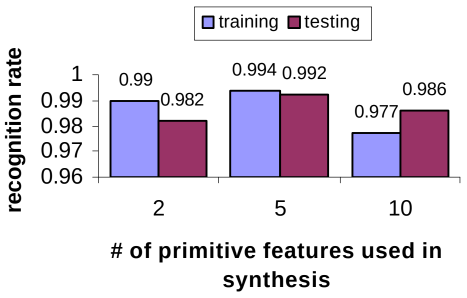


Figure 5.6. Experimental results with 5 sub-populations.

vectors are better than all the above results. Figure 5.7 shows the composite operator vector evolved by CGP maintaining 3 sub-populations in the 6th run when 5 primitive features are used, where PF*i* means the primitive feature *i* and so on.

(MULC (MULC (SUBC (SQRT (LOG PF8)))))

(a) Composite operator 1

(DIV (DIVC (DIVC (DIV PF18 PF6))) PF8)

(b) Composite operator 2

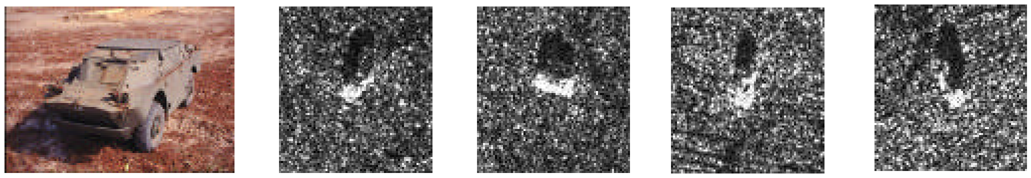
(SQRT PF8)

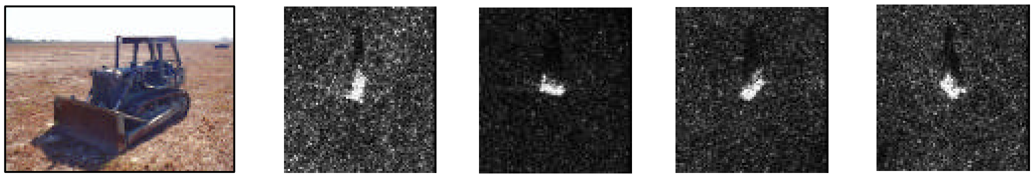
(c) Composite operator 3

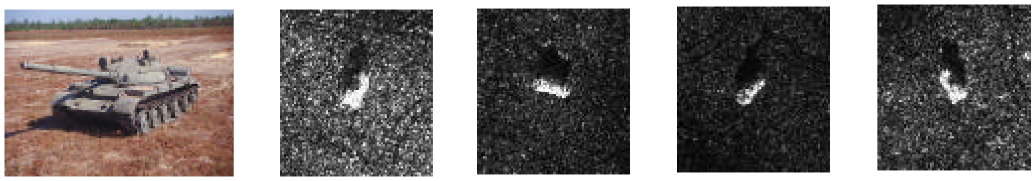
Figure 5.7. Composite operator vector learned by CGP.

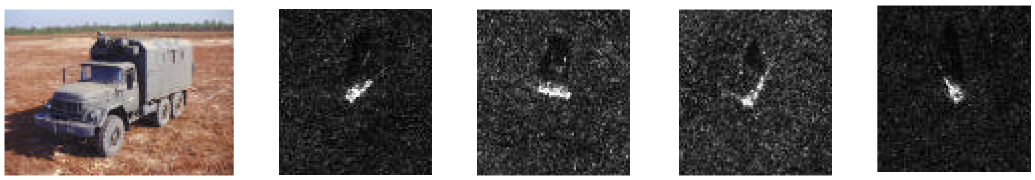
5.4.2 Recognize Objects

- **Data:** Five objects (BRDM2 truck, D7 bulldozer, T62 tank, ZIL131 truck and ZSU anti-aircraft gun) are used in the experiments. For each object, 210 real SAR images under 15°-depression angle and various azimuth angles between 0° and 359° are collected from MSTAR public data. Figure 5.8 shows one optical and four SAR images of each object. From Figure 5.8, we can see that it is not easy to distinguish SAR images of different objects. Since SAR images are very sensitive to azimuth angles and training images should represent the characteristics of an object under various azimuth angles, 210 SAR images of each object are sorted in the ascending order of their azimuth angles and the first, fourth, seventh, tenth SAR images and so on are selected for training. Thus, for each object, 70 SAR images are used in training and the rest are used in testing.


(a) Optical and SAR images of BRDM2.


(b) Optical and SAR images of D7.


(c) Optical and SAR images of T62.


(d) Optical and SAR images of ZIL.

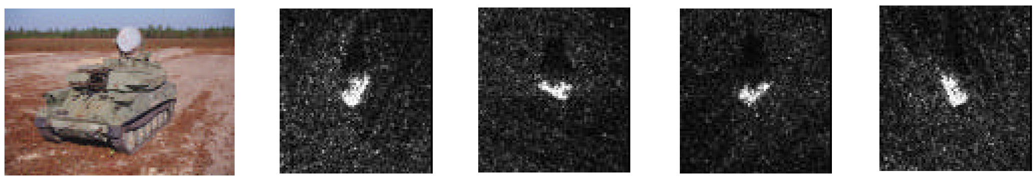

(e) Optical and SAR images of ZSU.

Figure 5.8. Five military objects used in recognition.

- **Experiment 2 - Discriminate three objects:** CGP synthesizes composite features to recognize three objects: BRDM2, D7 and T62. First, the efficacy of each primitive feature in discriminating these three objects is examined. The results are shown in Table 5.6 and Figure 5.9. Feature means values of pixels within blob (8) are the best primitive feature with recognition rate 0.73. Three series of experiments are performed in which CGP maintains 3, 5 and 8 sub-populations to evolve 3, 5 and 8-dimensional composite features, respectively. The primitive features used in the experiments are all the 20 primitive features and the last 10 primitive features. The experimental results are shown in Tables 5.7, 5.8 and 5.9. Figures 5.10, 5.11 and 5.12 show the average performance, where 10f and 20f mean primitive features 11 to 20 and all the 20 primitive features, respectively. The bins on the left show the training results and those on the right show the testing results. The numbers above the bins are the average recognition rates over all ten runs.

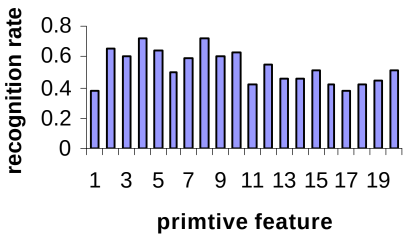


Figure 5.9. Recognition rates of 20 primitive features.

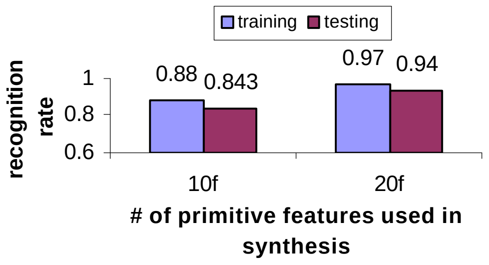


Figure 5.10. Recognition rates with 3 sub-populations.

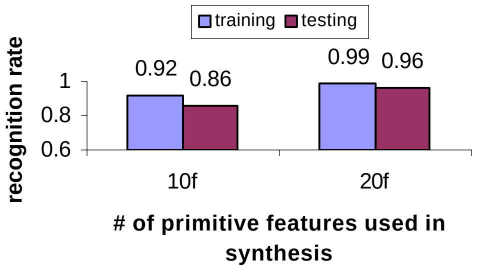


Figure 5.11. Recognition rates with 5 sub-populations.

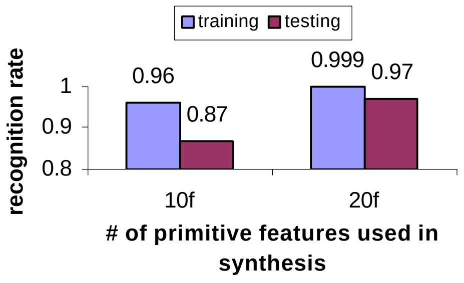


Figure 5.12. Recognition rates with 8 sub-populations.

From Figures 5.10, 5.11 and 5.12, it is clear that the learned composite feature vectors are more effective than primitive features. If all the 20 primitive features from the training images are used to form 20-dimensional primitive feature vectors to train a Bayesian classifier for recognition, the recognition rate is 0.96 on the testing images. This result is a little bit better than the average performance shown in Figure 5.10 (0.94), but the dimension of the feature vector is 20. However, the dimensions of composite feature vectors in Figures 5.10 and 5.11 are just 3 and 5 respectively. If the dimension of composite feature vector is increased from 3 to 5 and 8, the CGP results are better. If the last 10 primitive

Table 5.6. Recognition rates of 20 primitive features.

Feature Number	Primitive Feature	Recognition Rate	Feature Number	Primitive Feature	Recognition Rate
1	Standard deviation	0.376	11	Horizontal projection	0.414
2	Fractal dimension	0.662	12	Vertical projection	0.545
3	Weight-rank fill ratio	0.607	13	Major diagonal projection	0.460
4	Mass	0.717	14	Minor diagonal projection	0.455
5	Diameter	0.643	15	Minimum distance	0.505
6	rotational inertia	0.495	16	Maximum distance	0.417
7	Maximum CFAR	0.588	17	Mean distance	0.376
8	Mean CFAR	0.726	18	Moment μ_{20}	0.421
9	Percent bright CFAR	0.607	19	Moment μ_{02}	0.443
10	Count	0.633	20	Moment μ_{22}	0.512

Table 5.7. Experimental results with 3 sub-populations.

Runs	Recognition Rate			
	10f		20f	
	Training	Testing	Training	Testing
1	0.895	0.836	0.981	0.967
2	0.886	0.829	0.967	0.948
3	0.867	0.848	0.952	0.921
4	0.910	0.831	0.957	0.945
5	0.857	0.843	0.962	0.926
6	0.910	0.843	0.967	0.936
7	0.852	0.862	0.981	0.938
8	0.871	0.838	0.976	0.936
9	0.876	0.860	0.967	0.952
10	0.871	0.843	0.981	0.956
Average	0.880	0.843	0.969	0.943

Table 5.8. Experimental results with 5 sub-populations.

Runs	Recognition Rate			
	10f		20f	
	Training	Testing	Training	Testing
1	0.919	0.864	0.990	0.969
2	0.914	0.860	0.981	0.955
3	0.914	0.845	0.995	0.964
4	0.943	0.852	0.990	0.943
5	0.910	0.845	0.990	0.962
6	0.919	0.843	0.990	0.952
7	0.919	0.881	0.995	0.960
8	0.919	0.874	0.986	0.967
9	0.919	0.845	0.986	0.960
10	0.929	0.857	0.995	0.974
Average	0.921	0.857	0.990	0.961

Table 5.9. Experimental results with 8 sub-populations.

Runs	Recognition Rate			
	10f		20f	
	Training	Testing	Training	Testing
1	0.976	0.888	1.0	0.967
2	0.962	0.860	1.0	0.971
3	0.952	0.9	1.0	0.976
4	0.967	0.876	1.0	0.964
5	0.967	0.871	1.0	0.967
6	0.952	0.845	0.995	0.979
7	0.967	0.852	1.0	0.979
8	0.957	0.874	0.995	0.95
9	0.957	0.867	1.0	0.964
10	0.967	0.869	0.995	0.983
Average	0.962	0.870	0.999	0.97

```
(DIV (MULC (SUB (SUB (DIVC
(SQRT PF6)) (MULC (SUB PF18
(MULC (SUB PF18 (SQRT PF4))))))
(SQRT PF6))) (MIN2 PF12 PF19))
```

(a) Composite operator 1

```
(DIV (MULC (ADD (ADDC (MULC
(MUL (MIN2 (ADDC (DIV PF20
PF4)) PF14) PF3))) (LOG (ADDC
(DIV PF20 PF4))))) (DIVC PF4))
```

(b) Composite operator 2

```
(DIV (MIN2 (SUBC
(SUBC PF11)) (MAX2
PF7 PF8)) PF8)
```

(c) Composite operator 3

```
(PF11)
```

(d) Composite operator 4

```
(LOG (ADDC (LOG (DIV (SUBC
(LOG (DIV (SUBC (LOG PF5))
(SUBC PF5)))) (MUL PF2 PF5)))))
```

(e) Composite operator 5

Figure 5.13. Composite operator vector learned by CGP with 5 sub-populations.

features are used, the recognition rate is 0.81. From these results, we can see that the effectiveness of the primitive features has an important impact on the effectiveness of the composite features synthesized by CGP. With effective primitive features, CGP will synthesize better composite features. Figure 13 shows the composite operator vector evolved by CGP with 5 sub-populations in the 10th run using 20 primitive features. The size of the first and second composite operators is 20. The size of the third one and last one are 9 and 15, respectively. The fourth composite operator is just primitive feature 11. The primitive features used by the learned composite operator vector are primitive features 2, 3, 4, 5, 6, 7, 8, 11, 12, 14, 18, 19, 20. If all these 13 primitive features form 13-dimensional primitive feature vectors for recognition, the recognition rate is 0.96

- **Experiment 3 – Discriminate five objects:** With more objects added, the recognition becomes more difficult. This can be seen from Table 5.10 and Figure 14, which show the efficacy of each primitive feature in discriminating these five objects. Feature blob mass (4) is the best primitive feature with recognition 0.49. If all the 20 primitive features from the training images are used jointly to form 20-dimensional primitive feature vectors to train a Bayesian classifier for recognition, the recognition rate is 0.81 on the testing images; if only the last 10 primitive features are used, the recognition rate is 0.62. This number is much lower, since the last 10 features are common features and are not designed with the characteristics of SAR images taken into consideration.

Table 5.10. Recognition rates of 20 primitive features.

Feature Number	Primitive Feature	Recognition Rate	Feature Number	Primitive Feature	Recognition Rate
1	Standard deviation	0.224	11	Horizontal projection	0.273
2	Fractal dimension	0.473	12	Vertical projection	0.343
3	Weight-rank fill ratio	0.361	13	Major diagonal projection	0.281
4	Mass	0.486	14	Minor diagonal projection	0.265
5	Diameter	0.404	15	Minimum distance	0.277
6	rotational inertia	0.346	16	Maximum distance	0.294
7	Maximum CFAR	0.379	17	Mean distance	0.266
8	Mean CFAR	0.471	18	Moment μ_{20}	0.277
9	Percent bright CFAR	0.449	19	Moment μ_{02}	0.267

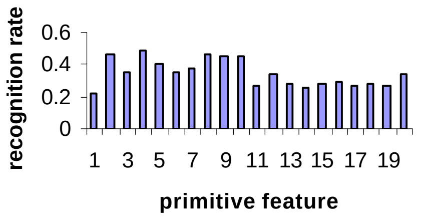


Figure 5.14. Recognition rates of 20 primitive features.

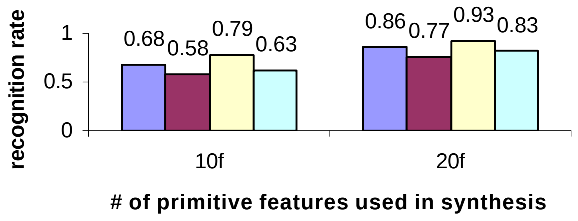


Figure 5.15. Recognition rates with 5 (left two bins) and 8 (right two bins) sub-populations.

Two series of experiments are performed in which CGP maintains 5 and 8 sub-populations to evolve 5 and 8-dimensional composite features for recognition. The primitive features used in the experiments are all the 20 primitive features and the last 10 primitive features. The maximum size of composite operators is 20. The experimental results are shown in Tables 5.11 and 5.12. Figure 5.15 shows the average recognition performance. The left two bins in columns 10f and 20f correspond to 5 sub-populations and the right two bins correspond to 8 sub-populations. The bins showing the training results are to the left of those showing the testing results. The numbers above the bins are the average recognition rates over all ten runs.

From Figure 5.15, we can see that when the dimension of the composite feature vector is 8, the performance of the composite features is good and it is better than using all 20 (0.81) or 10 (0.62) primitive features upon which the composite features are built. When the dimension of the composite feature vector is 5, the recognition is not satisfactory when using just 10 common features as building blocks. Also, when the dimension is 5, the average performance is a little bit worse than using all 20 or 10 primitive features, but the dimension of the composite feature vector is just one-fourth or half of the number of primitive

Table 5.11. Experimental results with 5 sub-populations.

Runs	Recognition Rate			
	10f		20f	
	Train-ing	Test-ing	Train-ing	Test-ing
1	0.691	0.594	0.84	0.727
2	0.674	0.581	0.831	0.741
3	0.706	0.627	0.866	0.75
4	0.697	0.594	0.846	0.756
5	0.666	0.563	0.869	0.777
6	0.671	0.549	0.863	0.787
7	0.654	0.546	0.829	0.747
8	0.683	0.577	0.837	0.753
9	0.691	0.573	0.874	0.794
10	0.674	0.587	0.88	0.803
Aver-age	0.681	0.579	0.854	0.764

Table 5.12. Experimental results with 8 sub-populations.

Runs	Recognition Rate			
	10f		20f	
	Train-ing	Test-ing	Train-ing	Test-ing
1	0.769	0.621	0.929	0.824
2	0.791	0.63	0.926	0.816
3	0.794	0.63	0.929	0.803
4	0.791	0.624	0.941	0.845
5	0.774	0.614	0.931	0.809
6	0.774	0.647	0.911	0.809
7	0.794	0.64	0.903	0.832
8	0.774	0.623	0.943	0.842
9	0.754	0.641	0.94	0.847
10	0.763	0.631	0.909	0.823
Aver-age	0.778	0.630	0.926	0.825

features, saving a lot of computational burden in recognition. When all the 20 primitive features are used and CGP has 8 sub-populations, the composite operators in the best composite operator vector evolved have sizes 19, 1, 16, 19, 15, 7, 16 and 6, respectively and they are shown in Figure 5.16. The primitive

(MIN2 PF10 (MIN2 (MULC (MUL PF9 (MIN2 (DIVC PF10) (MUL PF9 (DIVC PF10))))) (MIN2 (MUL PF9 (DIVC PF10)) PF10)))

(a) Composite operator 1.

(PF3)

(b) Composite operator 2.

(SUB (SUBC (SUB PF14 PF18)) (MAX2 (MAX2 (MAX2 PF14 PF8) PF14) (MAX2 PF14 (MAX2 PF14 PF5))))

(c) Composite operator 3.

(SQRT (DIV PF10 (SQRT (MAX2 (MULC (SUBC (DIV PF5 PF5))) (MAX2 (SUBC (MULC (SUBC (MULC (DIV PF15 PF5))))) PF10)))))

(d) Composite operator 4.

(LOG (MUL (LOG (SUB (ADD PF16 (SQRT (LOG (MUL (ADD PF16 PF16) PF12)))) PF12)) PF20))

(e) Composite operator 5

(SUB (LOG (DIVC PF2)) (DIV PF9 PF16))

(f) Composite operator 6.

(ADDC (ADD PF18 (ADD (MULC (LOG PF18)) (MIN2 (SUB PF2 PF11) (SUB PF18 (SUB PF11 PF2))))))

(g) Composite operator 7.

(SQRT (SQRT (SQRT (SQRT (SQRT PF4)))))

(h) Composite operator 8.

Figure 5.16. Composite operator vector learned by CGP.

features used by the synthesized composite operator vector are primitive features 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19 and 20. If all these 16 primitive features from the training images directly form 16-dimensional primitive feature vectors to train a Bayesian classifier for recognition, the recognition rate is 0.80 on the testing images, which is lower than the average performance of the composite feature vector shown in Figure 5.15.

5.4.3 Discussions

The above experiments show that CGP is a viable tool to synthesize effective composite features from primitive features for object recognition and the learned composite features outperform the primitive features or any combination of primitive features upon which they are evolved. The effectiveness of composite features learned by CGP is dependent on the effectiveness of primitive features. The usefulness of CGP is that it can evolve composite features that are more effective than the primitive ones upon which they are evolved. To achieve the same recognition rate, the number of composite features needed is smaller than the number of primitive features needed (one-fourth or half), thus, reducing the computational expenses during run-time recognition.

5.5 Summary

This chapter investigates synthesizing composite features for object recognition. Our experimental results using real SAR images show that CGP can evolve composite features that are more effective than the primitive features upon which they are built. To achieve the same recognition performance of primitive features, fewer composite features are needed and this reduces the computational burden during recognition. However, primitive features still have a significant impact on the effectiveness of the evolved composite features. How to let CGP evolve effective composite features using general primitive features is the focus of subsequent chapters.

Chapter 6

Feature Synthesis for Recognition Using Linear Genetic Programming

In this chapter, we present a novel method for learning complex concepts/hypotheses directly from raw training data. The task addressed here concerns data-driven synthesis of recognition procedures for real-world object recognition task. The method uses linear genetic programming to encode potential solutions expressed in terms of elementary operations, and handles the complexity of the learning task by applying cooperative coevolution to decompose the problem automatically. The training consists in coevolving feature extraction procedures, each being a sequence of elementary image processing and feature extraction operations. Extensive experimental results show that the approach attains competitive performance for 3-D object recognition in real synthetic aperture radar (SAR) imagery.

6.1 Introduction

Visual learning is a challenging domain for machine learning (ML) for several reasons.

- Firstly, visual learning is a *complex* task, that usually requires problem decomposition, which is nontrivial in itself.
- Secondly, the visual training data is *represented* in a way that is inconvenient for most standard ML methods, and requires use of specialized procedures and operators to access, aggregate, and transform the input.
- Thirdly, the *amount of data* that have to be processed during the training process is usually much higher than in standard ML applications. This imposes significant constraints on the effectiveness of the hypothesis space search.
- Finally, the real-world image data is usually *noisy* and contains plenty of *irrelevant* components that have to be sieved out in the learning process.

The approach for recognizing objects in real-world images described in this chapter addresses *all* these issues and attempts to solve these problems by using important ideas from machine learning, evolutionary computation (EC), and computer vision (CV), and combining them in a novel way.

6.2 Motivation, Related Work and Contribution

6.2.1 Motivation

The primary motivation for the research described in this chapter is the lack of general methodology for the design and development recognition systems. The design of recognition system for most real-world tasks is tedious, time-consuming and expensive. Though satisfactory in performance in constrained situations, the handcrafted solutions are usually limited in scope of applicability and have poor adaptation ability in practical applications. As the complexity of the task of object recognition by computer increases, the above limitations become severe obstacles for the development of solutions to real-world problems. In some aspects, it is similar to the way the complexity of the software development process made the developers struggle until the software engineering came into being.

6.2.2 Related Work

The interest in visual learning research has been rather limited in both ML and CV communities, although the importances of vision in the development of intelligent systems have been well recognized. In most approaches reported in the literature, adaptation is limited to parameter optimization that usually concerns a particular processing step, such as image segmentation, feature extraction, etc. In those cases, learning does affect the overall recognition result in some complex manner.

Current recognition systems are mostly open-loop and human input in the design of these systems is still predominant. Only a few contributions, summarized in Table 1, attempt to close the feedback loop of the learning process at the highest (e.g., recognition) level *and* test the proposed approach in real-world setting. Note that, to the best of our knowledge, only few approaches [60,53,55,49] have been reported that learn using *raw* images as training data, and, therefore, produce the *entire* object recognition system. Moreover, a majority of these methods [59, 48, 50] use domain-specific knowledge and are highly specialized towards a particular application.

Table 6.1. Related work in visual learning.

Reference	Approach	Experimental task	Training data
(Draper, 1993)	Learning recognition graphs	Recognizing buildings	Higher-level CV concepts
(Segen, 1994)	Learning of object models	Hand gesture recognition	Graphs extracted from images
(Johnson, 1995)	EC (GP)	Locating hand in human body silhouette	Binary silhouettes
(Teller & Veloso, 1997)	EC (GP variant)	Face recognition	Raw images (grayscale)
(Peng & Bhanu, 1998a)	Reinforcement learning	Segmentation of in/outdoor scenes	Raw images (color)
(Peng & Bhanu, 1998b)	Delayed reinforcement learning	Segmentation and feature extraction, in/outdoor scenes	Raw images (color)
(Krawiec, 2001)	EC (GP)	Handwriting recognition	Raw images (grayscale)
(Rizky et. al., 2002)	Hybrid EC (GP+NN)	Target recognition in radar modality	1-D radar signals
(Maloof et. al., 2003)	Standard ML/PR classifiers	Rooftop detection in aerial imagery	Fixed set of scalar features
This contribution	EC (CC+LGP)	Object recognition in radar modality	Raw image (grayscale)

6.2.3 Contributions

(a) We propose a *general approach to automatic learning/synthesis of recognition procedures*, that (i) uses raw image data for training, (ii) does not require domain-specific knowledge, and (iii) attains competitive performance on a complex, real-world object recognition task. The learning proceeds given only database of training examples (images) partitioned into decision classes, and a set of general-purpose image processing and feature extraction operators. We use the cooperative coevolution [57], a new paradigm of EC, to handle the complexity of the task.

(b) We use EC to perform the visual learning meant as the search in the space of image representations (features).

(c) We adopt a variety of linear genetic programming (LGP) [43] for encoding of basic image processing and feature extraction procedures.

(d) We use the real image data to demonstrate our approach and provide a comparison of performance between the coevolutionary approach and standard GA.

6.3 Technical Approach

The proposed approach operates in a learning-from-examples scheme, with learner/inducer autonomously acquiring knowledge from the training examples (images). The output of the learner is the synthesized recognition system that implements the *feature-based recognition* paradigm, with processing split into two stages: *feature extraction* and *decision making*. In particular, we include the image processing and feature extraction into the learning process (learning loop). The learner is, therefore, able to design the intermediate image representation that is appropriate for solving the task faced. Note that, from machine learning viewpoint, this approach may be regarded as a kind of constructive induction [51].

6.3.1 Evolving Recognition Procedures

The learning proceeds in the framework of evolutionary computation, where we evolve *procedures* being sequences of elementary image processing and feature extraction operations. The evolutionary algorithm maintains a set of such procedures that are modified and mated during the evolutionary search (Figure 6.1). The procedures compete with each other by means of their fitness values that reflect the utility of particular representation for solving the problem. The best procedure found in the evolutionary run becomes the final result of the procedure synthesis.

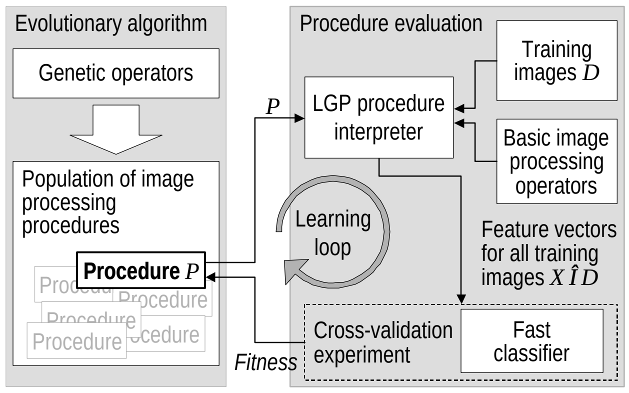

Evolutionary algorithm
Genetic operators
Population of image processing procedures
Procedure P
Procedure
Procedure
Procedure
Procedure
Procedure
P
Fitness
Procedure evaluation
LGP procedure interpreter
Training images D
Basic image processing operators
Learning loop
Feature vectors for all training images X Î D
Cross-validation experiment
Fast classifier

Figure 6.1. The overall architecture of our learning system.

6.3.2 Representation of Feature Extraction Procedures

An important issue that influences the performance of the proposed approach is the representation of individuals. To speed up the convergence of the search process and provide the system with basic knowledge, we assume that certain elementary building blocks are given *a priori* to the learner in a form of basic image processing, feature extraction, and feature transformation operators.

A variety of linear genetic programming (LGP) [43] is chosen as the representation framework for the described system. LGP is a hybrid of genetic algorithms (GA), and genetic programming (GP). The LGP *genome*, i.e. the internal encoding of solution, is a fixed-length string of numbers that is interpreted as a sequential procedure. The procedure is composed of (possibly parameterized) basic operations that work on input data/images. The major advantage of this linear representation is low susceptibility to destructive crossovers, which is an important problem in GP.

The details of LGP procedure encoding may be briefly summarized as follows:

- Each procedure *P* is a fixed-length string of bytes [0.255] that encodes sequence of *operators,* i.e. image processing and feature extraction algorithms.
- The operations work on *registers* (working variables) used for both input and output during procedure execution. *Image registers* store processed images, whereas *real-number registers* store scalar features. All image registers have the same dimensions as the input image. Each image register, apart from storing the image, maintains a single rectangular mask. A single learning parameter n_{reg} controls both the number of image and number registers.
- Each chunk of 4 consecutive bytes in the LGP procedure encodes a single operation with the following elements: (i) operation code, (ii) mask flag – decides whether the operation should be global (work on the entire image) or local (limited to the mask), (iii) mask dimensions (ignored if mask flag is 'off'), (iv) arguments – numbers (identifiers) of registers to fetch input data and store the result.

An example of operation is morphological opening (operation code) using rectangular ROI (ROI flag 'on') of size 14 (ROI size) on the image fetched from image register #4 (pointed by argument #1), and storing the result in image register #5 (pointed by argument #2).

There are currently approximately 70 operations implemented in the system, consisting mostly of Intel Image Processing [46] and OpenCV [47] libraries. They may be grouped into following categories: image processing operations, mask – related operations, feature extraction operations, and arithmetic and logic operations.

Given the above settings, an LGP procedure *P* processes a single input image *I* in following steps (see Figure 6.2):

1. *Initialization* of register contents: Each of the n_{reg} image registers is set to *I*. The masks of images are set to consecutive local features (here: bright 'blobs') found in the image, so that mask in the i^{th} image register encompasses i^{th} local feature. Real-number registers are set to the midpoint coordinates of corresponding masks; in particular, real-number registers $2i$ and $2i+1$ store the x and y coordinates of the i^{th} image mask.

2. *Execution*: the operations encoded by P are carried out one by one. As a result, the contents of image and real-number registers change (see example in Figure 6.2).
3. *Interpretation*: the values computed and stored in the real-value registers are interpreted as the output yielded by P for image I. Let us denote by $f_i(P,I)$ the value stored by P in the real-value register #i when processing image I. Then, for an image I, the LGP procedure outputs a vector of features:

$$\left\langle f_1(P,I), f_2(P,I), \ldots, f_{n_{reg}}(P,I) \right\rangle$$

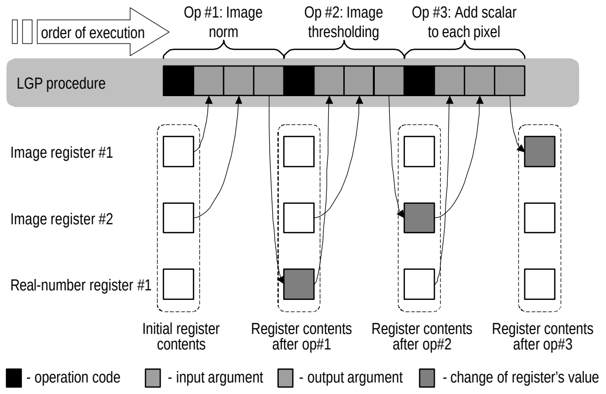

order of execution
Op #1: Image norm
Op #2: Image thresholding
Op #3: Add scalar to each pixel
LGP procedure
Image register #1
Image register #2
Real-number register #1
Initial register contents
Register contents after op#1
Register contents after op#2
Register contents after op#3
- operation code
- input argument
- output argument
- change of register's value

Figure 6.2. Illustration of the process of genome interpretation during LGP procedure execution.

6.3.3 Cooperative Coevolution

To cope with the *inherent complexity* of the visual learning task, we should find a way to *decompose the problem* into subtasks rather than trying to solve it in one step. For that purpose, we use the cooperative coevolution, a variety of evolutionary computation.

Evolutionary computation is widely recognized as a kind of *metaheuristics*, i.e. general-purpose search algorithm that provides suboptimal solutions in polynomial time. However, according to Wolpert's 'no free lunch' theorem [63], the search for an universal, best-of-all metaheuristic (optimization or learning) algorithm is futile. In other words, the average performance of any metaheuristic over a set of all possible fitness functions is the same.

In real world however, not all fitness functions are equally probable. Most real problems are characterized by some *features* that make them specific. The practical utility of a search/learning algorithm depends, therefore, on its ability to detect and benefit from that specificity. In particular, the *complexity* of the problem and the way it may be *decomposed* are such characteristics.

In the last few years, *cooperative coevolution* (CC) [56], a variety of EC, has been reported as a promising approach to handle the increasing complexity of problems posed in artificial intelligence and related disciplines. There are two important factors that make CC different from standard EC. Firstly, instead of having just one population of individuals, in CC one maintains *many* of them. Secondly, individuals in particular population encode only *part* of the solution to the problem, as opposed to EC, where each individual encodes complete solution to the problem. Therefore, individuals from populations cannot be evaluated independently; they have to be combined with some *representatives* from the remaining populations to form a solution that can be evaluated. That is why evolution proceeds here in each population independently, with the exception of the evaluation stage. The joint evaluation scheme forces the individuals from particular populations to cooperate.

Let n denote the number of populations. To evaluate an individual X from ith population (Figure 6.3), it is temporarily combined with selected individuals (so called *representatives*) from the remaining populations j, j=1,…, n, j^1i, to form the solution. Then, the entire solution is evaluated by means of the fitness function and X gets the resulting fitness value. Evaluation of an individual from ith population does not affect the remaining populations. As a result, the evolutionary search in a given population is driven by the context build up by the representatives of remaining populations. The choice of representatives is, therefore, critical for the convergence of the evolution process. Although many different variants are possible here, it has been shown that so-called CCA-1 scheme works best [61]. In the first generation a representative of ith population is an individual drawn randomly from it. In the following generations a representative of ith population is the best individual w.r.t. the previous generation.

```
initialize populations
loop
    for each population
        for each individual X
            combine X with representatives of
                remaining populations to form solution S
            evaluate S
            assign fitness of S to X
        end for
        select mating candidates
        recombine parents and use their offspring as the
    next
        generation
    end for
until stopping condition
return best solution
```

Figure 6.3. Outline of cooperative coevolution algorithm.

The major advantage of CC is that it provides the possibility of breaking up a complex problem into components *without specifying explicitly the objectives for them*. The way the individuals from populations cooperate *emerges* as the evolution proceeds. In [44] we provided experimental evidence for the usefulness of CC in feature construction for standard machine learning problems. Here we claim that CC is especially appealing also to the problem of visual learning, where the overall target is well defined, but there is no *a priori* knowledge about what should be expected at intermediate stages of processing, or such knowledge requires an extra effort from the designer.

6.3.4 Combining Cooperative Coevolution and Linear Genetic Programming

In the proposed approach, we use cooperative coevolution to scale down the task of LGP procedure synthesis (Section 6.2). Although this can be done in many different ways, in this initial contribution we break up the task at *genome level*, with each population being responsible for optimizing a pre-defined fragment (substring) of LGP code of fixed length (Figure 6.4).

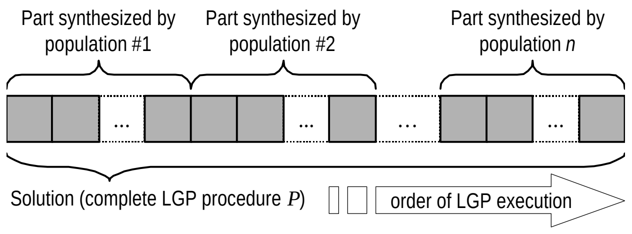

Part synthesized by population #1
Part synthesized by population #2
Part synthesized by population *n*
Solution (complete LGP procedure *P*)
order of LGP execution

Figure 6.4. Cooperation enforced by the concatenation of LGP procedure fragments developed by particular populations.

The evaluation of an individual *X* from a given population consists in concatenating (always in the same order) its genome with the genomes of the representatives of the remaining populations to form a single LGP procedure *P*. *P* is then executed for all images from the training set (see Section 6.2). The values computed by *P* for all training images

$$\left\langle f_1(P,I), f_2(P,I), \dots, f_{n_{reg}}(P,I) \right\rangle, \forall I \in T,$$

together with the images' class labels constitute the dataset *T* that is the basis for evaluation of an individual (so-called *fitness set*). Then, a fast classifier is trained and tested on these data (see Figure 6.1), using predefined internal division of the training set into training-training set and training-testing set. For this purpose, we used the naïve Bayesian classifier, modeling the input variables (features) by normal distribution. The resulting predictive *recognition ratio*,

$$\frac{\#\text{ of correctly classified objects from } T}{\text{total }\#\text{ of images in } T},$$

becomes the evaluation (fitness) of the solution-procedure *P*, and is subsequently assigned to the individual *X*.

In this framework, particular populations can specialize in different stages of the recognition task. In particular, we expect that the populations delegated to the development of the early parts of LGP procedure would tend to specialize in image processing, whereas the populations working on the final parts of the LGP procedure would focus on feature extraction and aggregation.

6.4 Experiments

The objective of the computational experiments is to explore the overall idea of LGP-based synthesis of recognition procedures using cooperative coevolution for search, in the context of demanding, real-world object recognition task using images of 3-D objects. The results are obtained using a PC with single Pentium 1.8 GHz processor.

To provide a reference solution, we run a separate series of standard linear genetic programming (LGP), which, in fact, is a special case of CC that uses just one population. To make this comparison reliable, we *fix the total genome length* (the total procedure length is the same for both CC and standard LGP), and *fix the total number of individuals* (the total number of individuals from all populations in CC is equal to the number of individuals maintained in the single population of the corresponding LGP run). To estimate the performance the learning algorithm is able to attain in a limited time, evolution stops when its run time reaches the predefined limit.

6.4.1 Parameter Setting

Table 6.2. Parameter setting.

Parameter	*Setting*
Mutation operator	one-point, prob. 0.5
Crossover operator	one-point, prob. 1.0, genome cutting is allowed at every point
Selection operator	tournament selection with tournament pool size = 5
Number or registers (image and numeric) n_{reg}	8
Number of populations n	3
Selection of representatives	CCA-1 (see Section 3.3)
Time limit	1000 and 2000 seconds
Procedure length (total genome length)	72 bytes, i.e., 18 operations
Total population size	300 - 900 individuals

Table 6.2 shows the details of parameter settings used for the experiments described in this section. All the remaining parameters were set to default values used in software packages ECJ [50] and WEKA [62].

6.4.2 Data and the Learning Task

The proposed approach has been tested on the demanding task of object recognition in synthetic aperture radar (SAR) imagery. The MSTAR public database [58] of SAR images taken at one foot resolution has been used as the data source. The task posed to the system was to recognize three different objects (decision classes): BRDM2, D7, and T62 (see Figure 6.4) at 15° depression angle and any azimuth (0°-359°).

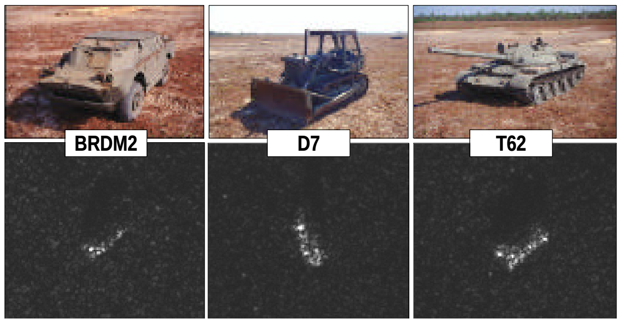


Figure 6.5. The representatives of three decision classes. Top row – visual photographs, bottom row - corresponding 48×48 pixel SAR images.

The difficulties associated with the object recognition task in real SAR images are:

- Non-literal nature of the data, i.e. radar images appear different than visual ones. Bright spots on the images, called scattering centers, correspond to those parts of the object which reflect radar signal strongly. No line features are present for these man-made objects at this resolution.
- Low persistence of features under rotation (high rotation-variance).
- High levels of noise.

Table 6.3. Dataset statistics.

		Number of images			
Class	*Total*	*Training set*	*Aspect interval*	*Testing set*	*Aspect interval*
BRDM2	188	64	5.62°	124	2.90°
D7	188	64	5.62°	124	2.90°
T62	131	64	5.62°	67	5.37°
Total	507	192		315	

Table 6.4. The average performances of best individuals evolved in 10 independent runs for 1000 and 2000 seconds training time limit.

Method	*# popu-lations*	*Parameter setting*				*Recognition ratio*			
		Procedure length		*# of individuals*		*1000 seconds*		*2000 seconds*	
		Each population	*Total*	*Each poplation*	*Total*	*Train set*	*Test set*	*Train Set*	*Test Set*
CC	3	24	72	100	300	0.915	0.867	0.933	0.890
LGP	1	72	72	300	300	0.806	0.747	0.843	0.801
CC	3	24	72	300	900	0.927	0.874	0.940	0.883
LGP	1	72	72	900	900	0.839	0.795	0.881	0.830

From the MSTAR database, 507 images of three objects classes (see Figure 6.5) have been selected. The resulting set of images has been split into disjoint training and testing parts to provide reliable estimate of the recognition ratio of the learned recognition system (see Table 6.3). This selection was aimed at providing uniform coverage of the azimuth (for each class, there is a training image for approximately every 5.62° of azimuth, and a testing image every 2.9°-5.37°, on the average).

The evolutionary process uses the training data for the learning/synthesis (precisely speaking, for the fitness computation), whereas the testing images are used for test only. The original images have different sizes, so they are cropped to 48×48 pixels. They are also complex (2-channel), but only theirs magnitude part is used in the experiments. No other form of preprocessing (e.g., speckle removal) is applied.

6.5 Results

Table 6.4 compares the recognition performances obtained by the proposed coevolutionary approach (CC) and its regular counterpart (LGP), for two different limits imposed on the evolutionary learning time, 1000 and 2000 seconds. To obtain statistical evidence, all evolutionary runs have been repeated 10 times, so the table presents the average performances of the best individuals found.

The direct comparison resulting from Table 6.4 shows the superiority of the CC to LGP. This applies to both the performance of the synthesized systems on the training as well as on the test set. In all cases, the observed increases in accuracy are statistically significant with respect to the one-sided t-Student test at the confidence level 0.05. Note that, within the same time limit, CC usually ran for a *smaller* number of generations on the average, due to the extra time required to maintain (perform selection and mating) in multiple populations.

Figure 6.6 and Table 6.5 show, respectively, the receiver operating characteristics (ROC) curves and confusion matrices for the best individuals found in the first two experiments reported in Table 6.4 (time limit: 2000 seconds, procedure length: 72, total # of individuals: 300). Each curve shows the true positive ratio, i.e., the share of correctly recognized objects, as a function of false positive ratio, i.e., the share of incorrectly classified objects (without taking into account the non-recognized objects).

These parametric characteristics have been obtained from the test set, by varying the confidence threshold of the naïve Bayesian classifier. Approximately 40 different values of the threshold have been used to obtain the curves. The confidence threshold imposes a lower limit on the ratio of *a posteriori* probabilities of the first and the second most probable decision classes. If, for a particular test example, the ratio is lower than threshold, no recognition decision is made and the example remains unclassified. The ROC curves clearly show the superiority of the coevolution. For instance, when no more than 5% of false positives are allowed, the procedure evolved using CC recognizes correctly approximately 91% images, whereas for LGP the accuracy is around 68%.

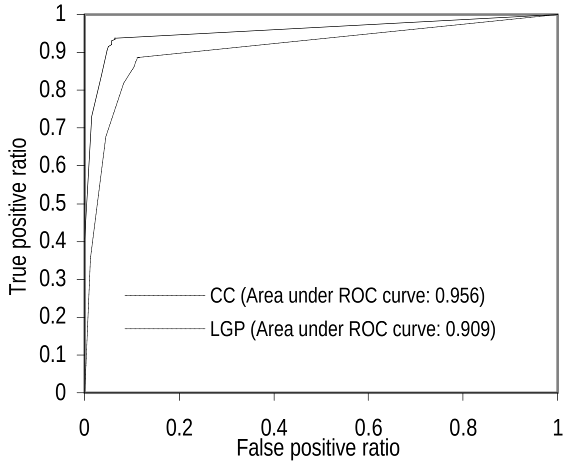


Figure 6.6. ROC curves obtained for the test set using the best individuals found in the first two experiments shown in Table 6.4.

Figure 6.7 presents the processing carried out on a BDRM2 image (taken at 342.3° aspect) by the best procedure found in one of evolutionary runs. For clarity, the picture shows the interpretation of the LGP procedure in a form of data-flow graph. Note that this procedure uses only first four of the total of eight image registers available. Each column of images in the picture shows the content changes of particular image register.

The execution of the LGP procedure starts from the top and proceeds downwards through several intermediate image-processing steps. Rounded and slanted boxes denote global (working on the entire image) and local (working on the marked rectangular ROI mask) image processing operations, respectively. Eventually, two of the executed operations yield scalar features (the *x* coordinate of the shifted ROI ($f_1(X,I)$), and the normalized difference of two processed images

$f_2(X,I)$). The overall processing ends with the final recognition decision made by the (previously trained on the training set) classifier; this includes *a posteriori* probabilities yielded by the naïve Bayesian classifier.

The operations used in this particular are: *AbsDiff* – pixel-wise absolute difference of a pair of images, *HiPass3x3* – high pass convolution filter using 3×3 mask, *CrossCorrel* – cross-correlation of a pair of images, *PushROIX* – (local) shifts the current image's ROI to the closest bright 'blob' in horizontal direction, *Gaussian* – (local) image smoothing using 3×3 Gaussian mask, *MorphClose* – morphological closing operation, *LogicalOr* – pixel-wise logical 'OR' operation.

Table 6.5. Confusion matrices for the test set using the best individuals found in the first two experiments shown in Table 6.4.

CC		*Predicted class*		
Actual class	*BRDM2*	*D7*	*T62*	*None*
BRDM2	**118**	1	4	1
D7	5	**114**	3	2
T62	5	1	**61**	0

LGP		*Predicted class*		
Actual class	*BRDM2*	*D7*	*T62*	*None*
BRDM2	**97**	3	22	2
D7	0	**115**	9	0
T62	1	0	**66**	0

Note that, commonly for genetic programming, not all input data (initial register contents) and not all intermediate results are utilized for the final decision making (e.g., the result of the cross-correlation operation (*CrossCorrel*) is not further processed).

6.6 Summary

In this chapter, we proposed a general evolutionary learning method that enables the learner to *acquire knowledge from complex/structural examples by autonomously transforming the input representation*. The described formulation of feature construction addresses two important issues. (1) The elementary operations give the learner an access to complex, structural input data that otherwise could not be directly used. (2) By incorporating the feature synthesis into the learning loop, the learner searches for performance improvement by modifying the input representation.

In experimental part, we provided an evidence for the possibility of solving, using the proposed approach, a demanding real-world task of *visual learning*. The encouraging results for SAR object recognition have been obtained without recurring to means that are commonly used in conventional approaches to the design of recognition systems, such as resorting to the database of object models, explicit estimation of object pose, hand-tuning of basic operations for a specific application, and, in particular, SAR-specific concepts or features like 'scattering center'.

Our approach learns in a *fully automatic manner*, and, therefore, at a little expense of human labor and expertise. The learning process requires only training data that is usually easy to acquire, i.e. images and their class labels, and does not rely on domain-specific knowledge, using only general vision-related knowledge encoded in basic operations. The objectivity of the learning process makes the results free from subjective flaws and biases, which the human-designed solutions are prone to.

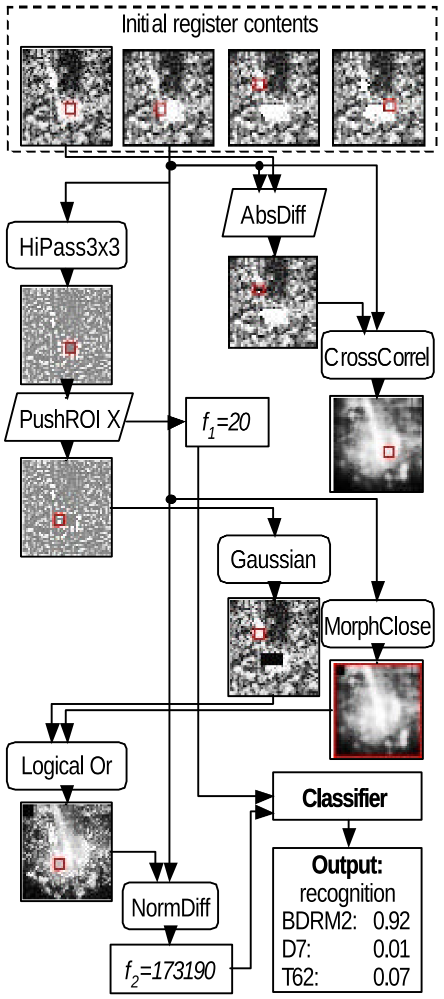


Figure 6.7. A fragment of synthesized processing graph of a selected best-of-run procedure evolved by means of cooperative coevolution, processing an exemplary image (only 4 of total 8 registers are used by this procedure).

The proposed method may be characterized as *feature-based*. Compared to the model-based recognition approaches, there is no need for, possibly expensive, matching an image with models from the database. Thus, our synthesized recognition system attains *high recognition speed* during the runtime. The average time required by the entire recognition process, starting from the raw image and ending up with the final recognition result, totaled 4.9 ms on the average, for a single 48×48 image and an LGP procedure composed of 18 operations. This time could be significantly reduced after re-implementing the synthesized system and, in particular, the classifier that is written in Java. We claim that this impressive recognition speed makes our approach suitable for real-time application.

Since the task-related knowledge is not required, our approach is general and possibly applicable to other recognition tasks. We claim that, therefore, a new paradigm for visual learning has been developed, that focuses on automatic learning of pattern analysis procedures composed of relatively simple, general-purpose image processing and feature extraction building blocks, as opposed to the tendency of designing highly specialized procedures for particular recognition tasks.

From machine learning viewpoint, this result is an outstanding argument in favor of CC for tackling complex learning problems. The ability of coevolution to break up complex problems into subproblems without requiring explicit objectives/goals for them, offers an interesting research direction for ML, when complex learning tasks are involved.

Chapter 7

Coevolution and Linear Genetic Programming for Recognition-Further Extensions

In this chapter, we provide further details and extend the approach presented in Chapter 6. Given the training images, this general approach induces a sophisticated feature-based recognition system, by using cooperative coevolution and linear genetic programming for the procedural representation of feature extraction agents. An extensive experimental evaluation, on the demanding real-world task of object recognition in synthetic aperture radar (SAR) imagery, shows the competitiveness of the proposed approach with human-designed recognition systems.

7.1 Introduction

Most real-world learning tasks concerning visual information processing are inherently complex. This complexity results not only from the large volume of data that one usually needs to process, but also from its spatial nature, information incompleteness, and, most of all, from the vast number of hypotheses that have to be considered in the learning process and the 'ruggedness' of the fitness landscape. Therefore, the design of a visual learning algorithm mostly consists in *modeling* its capabilities so that it is effective in solving the problem. To induce useful hypotheses on one hand and avoid overfitting to the training data on the other, some assumptions have to be made, concerning training data and hypothesis representation, known as *inductive bias* and *representation bias,* respectively. In visual learning, these biases have to be augmented by an extra '*visual bias*', i.e., knowledge related to the visual nature of the information being subject to the learning process. A part of that is general knowledge concerning vision (*background knowledge,* BK), for instance, basic concepts like pixel proximity, edges, regions, primitive features, etc. However, usually a more specific *domain knowledge* (DK) related to a particular *task/application* (e.g., fingerprint identification, face recognition, etc.) is also required. Currently, most recognition methods make intense use of DK to attain a competitive performance level. This is, however, a double-edged sword, as the more DK the method uses, the more specific it becomes and the less general and transferable is the knowledge it acquires. The contribution of such over-specific methods to the overall body of knowledge is questionable.

Therefore, in this chapter, we propose a general-purpose visual learning method that requires only BK and produces a complete recognition system that is able to classify objects in images. To cope with the complexity of the recognition task, we break it down into components. However, the ability to identify building blocks is a necessary, but not a sufficient, precondition for a successful learning task. To enforce learning in each identified component, we need an *evaluation function* that spans over the space of all potential solutions and guides the learning process. Unfortunately, when no *a priori* definition of module's 'desired output' is available, this requirement is hard to meet. This is why we propose to employ here cooperative coevolution [56], as it does not require the explicit specification of objectives for each component.

7.2 Related Work and Contributions

No general methodology has been developed so far that effectively automates the visual learning process. Several methods have been reported in the literature; they include blackboard architecture, case-based reasoning, reinforcement learning, and automatic acquisition of models, to mention the most predominant. The paradigm of evolutionary computation (EC) has also found applications in image processing and analysis. It has been found effective for its ability to perform global parallel search in high-dimensional search spaces and to resist the local optima problem. However, in most approaches the learning is limited to parameter optimization. Relatively few results have been reported [66, 53, 59, 60], that perform visual learning in the deep sense, i.e., with a learner being able to synthesize and manipulate an entire recognition system.

The major contribution of this chapter is a general method that, given only a set of training images, performs visual learning and yields a complete feature-based recognition system. Its novelty consists mostly in (i) *procedural representation* of features for recognition, (ii) utilization of *coevolutionary computation* for induction of image representation, and (iii) *a learning process* that optimizes the image feature definitions, prior to classifier induction.

7.2.1 Coevolutionary Construction of Feature Extraction Procedures

We pose visual learning as the search of the space of image representations (sets of features). For this purpose, we propose to use *cooperative coevolution* (CC) [56], which, besides being appealing from the theoretical viewpoint, has been reported to yield interesting results in some experiments [61]. In CC, one maintains many populations, with individuals in populations encoding only a *part* of the solution to the problem. To undergo evaluation, individuals have to be (temporarily) combined with individuals from the remaining populations to form an *organism* (solution). This joint evaluation scheme forces the populations to cooperate. Except for this evaluation step, other steps of evolutionary algorithm proceed in each population independently.

According to Wolpert's 'No Free Lunch' theorem [63], the choice of this particular search method is irrelevant, as the average performance of any metaheuristic search over a set of all possible fitness functions is the same. In the real world, however, not all fitness functions are equally probable. Most real-world problems are characterized by some features that make them specific. The practical utility of a search/learning algorithm depends, therefore, on its ability to detect and benefit from those features.

The *high complexity* and *decomposable nature* of the visual learning task are such features. Cooperative coevolution seems to fit them well, as it provides the possibility of breaking up a complex problem into components *without specifying explicitly the objectives for them*. The manner in which the individuals from populations cooperate *emerges* as the evolution proceeds. In our opinion, this makes CC especially appealing to the problem of visual learning, where the overall object recognition task is well defined, but there is no *a priori* knowledge about what should be expected at intermediate stages of processing, or such knowledge requires an extra effort from the designer.

In [44], we provide experimental evidence for the superiority of CC-based feature construction over standard EC approach in the standard machine learning setting; here, we extend this idea to visual learning. Following the *feature-based recognition* paradigm, we split the object recognition process into two modules: *feature extraction* and *decision-making*. The algorithm learns from a finite training set of examples (images) D in a *supervised* manner, i.e. requires D to be partitioned into finite number of pairwise disjoint decision classes D_i.

In the coevolutionary run, n populations cooperate in the task of building the complete image *representation*, with each population responsible for evolving one component. Therefore, the cooperation here may be characterized as taking place at the *feature level*. In particular, each individual I from a given population encode a single *feature extraction procedure*. For clarity, details of this encoding are provided in Section 7.4.

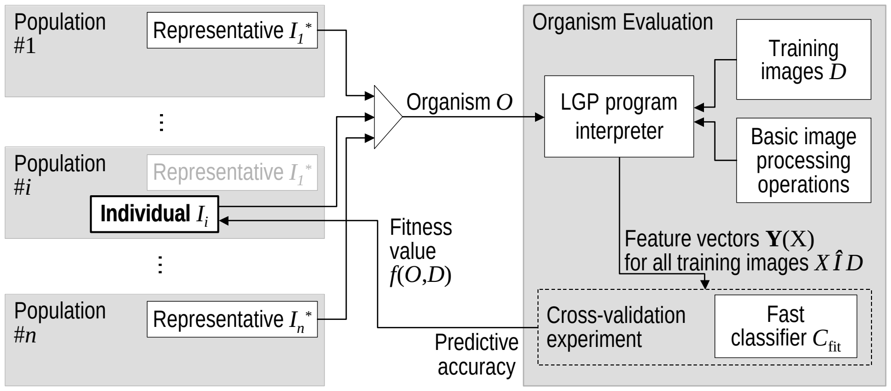


Figure 1. The evaluation of an individual I_i from i^{th} population.

The coevolutionary search proceeds in all populations independently, except for the evaluation phase, shown in Figure 1. To evaluate an individual I_j from population #*j*, we first provide for the remaining part of the representation. For this purpose, representatives I_i^* are selected from all the remaining populations $i \neq j$. A representative I_i^* of i^{th} population is defined here in a way that has been reported to work best [61]: it is the best individual w.r.t. the previous evaluation. In the first generation of evolutionary run, since no prior evaluation data is given, it is a randomly chosen individual.

Subsequently, I_j is temporarily combined with representatives of all the remaining populations to form an organism

$$O = \left\langle I_1^*, \ldots, I_{j-1}^*, I_j, I_{j+1}^*, \ldots, I_n^* \right\rangle . \tag{7.1}$$

Then, the feature extraction procedures encoded by individuals from *O* are 'run' (see Section 7.4) for all images *X* from the training set *D*. The feature values **y** computed by them are concatenated, building the compound feature vector **Y**:

$$\mathbf{Y}(X) = \left\langle \mathbf{y}(I_1^*, X), \ldots, \mathbf{y}(I_{j-1}^*, X), \mathbf{y}(I_j, X), \mathbf{y}(I_{j+1}^*, X), \ldots, \mathbf{y}(I_n^*, X) \right\rangle . \tag{7.2}$$

Feature vectors **Y**(*X*), computed for all training images $X \in D$, together with the images' decision class labels constitute the dataset:

$$\{\left\langle \mathbf{Y}(X), i \right\rangle : \forall X \in D_i, \forall D_i\} \tag{7.3}$$

Finally, cross-validation, i.e. multiple train-and-test procedure is carried out on these data. For the sake of speed, we use here a fast classifier C_{fit} that is usually much simpler than the classifier used in the final recognition system. The resulting predictive recognition ratio (see equation 4) becomes the evaluation of the organism *O*, which is subsequently assigned as the fitness value to *f* ( ) the individual I_j, concluding its evaluation process:

$$f(I_j, D) = f(O, D) = \\ = card\left(\left\{\left\langle \mathbf{Y}(X), i \right\rangle, \forall X \in D_i \wedge C(\mathbf{Y}(X)) = i, \forall D_i, \right\}\right) / \, card(D) \tag{7.4}$$

where *card*() denotes cardinality of a set. Using this evaluation procedure, the coevolutionary search proceeds until some stopping criterion (usually considering computation time) is met. The final outcome of the coevolutionary run is the best found organism/representation O^*.

7.2.2 Representation of Feature Extraction Procedures

For representing the feature extraction procedures as individuals in the evolutionary process, we adopt a variety of Linear Genetic Programming (LGP) [43], a hybrid of genetic algorithms (GA) and genetic programming (GP). The individual's genome is a fixed-length string of bytes, representing a sequential program composed of (possibly parameterized) basic *operations* that work on images and scalar data. This representation combines advantages of both GP and GA, being both procedural and more resistant to the destructive effect of crossover that may occur in 'regular' GP [43].

A feature extraction procedure accepts an image X as input and yields a vector $\mathbf{y}$ of scalar values as the result. Its operations are effectively calls to image processing and feature extraction functions. They work on *registers*, and may use them for both input as well as output arguments. *Image registers* store processed images, whereas *real-number registers* keep intermediate scalar results features. Each image register has single channel (grayscale), the same dimensions as the input image X, and maintains a rectangular *mask* that, when used by an operation, limits the processing to its area. For simplicity, the numbers of both types of registers are controlled by the same parameter m.

Each chunk of four consecutive bytes in the genome encodes a single operation with the following components:

(a) operation code,

(b) mask flag – decides whether the operation should be global (work on the entire image) or local (limited to the mask),

(c) mask dimensions (ignored if the mask flag is 'off'),

(d) arguments: references to registers to fetch input data and store the result.

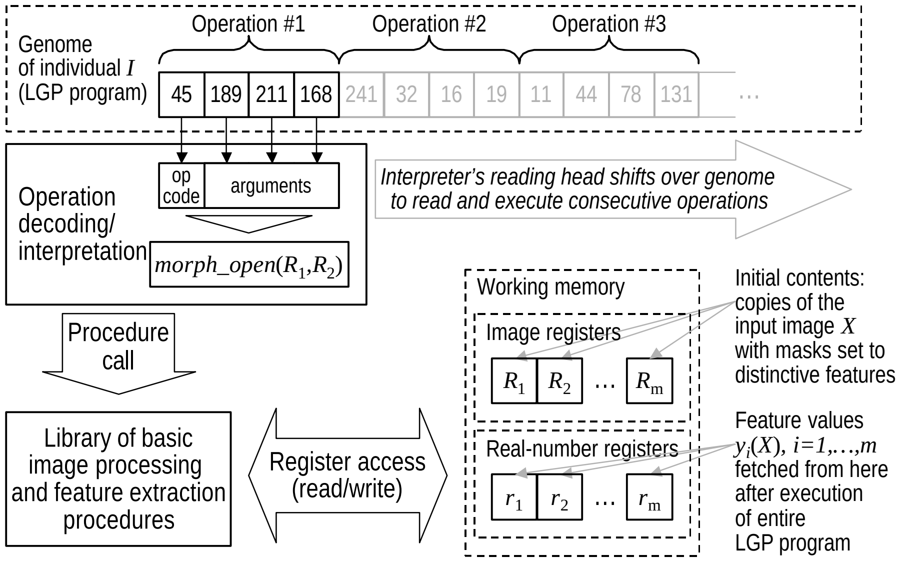

Operation #1
Operation #2
Operation #3
Genome of individual I (LGP program)
45 189 211 168 241 32 16 19 11 44 78 131 …
op code
arguments
Operation decoding/ interpretation
morph_open(R1,R2)
Interpreter's reading head shifts over genome to read and execute consecutive operations
Procedure call
Library of basic image processing and feature extraction procedures
Register access (read/write)
Working memory
Image registers
R1 R2 … Rm
Real-number registers
r1 r2 … rm
Initial contents: copies of the input image X with masks set to distinctive features
Feature values yi(X), i=1,…,m fetched from here after execution of entire LGP program

Figure 7.2. Execution of LGP code contained in individual's *I* genome (for a single image *X*).

Figure 7.2 shows the execution at the moment of executing the following operation: morphological opening (a), applied locally (b) to the mask of size 14×14 (c) to the image fetched from image register pointed by argument #1, and storing the result in image register pointed by argument #2 (d). There are currently 70 operations implemented in the system. They mostly consist of calls to functions from Intel Image Processing and OpenCV libraries, and encompass image processing, mask-related operations, feature extraction, and arithmetic and logic operations.

The processing of a single input image $X \in D$ by the LGP procedure encoded in an individual *I* proceeds as follows (Figure 7.2):

1. *Initialization*: Each of the *m* image registers is set to *X*. The masks of images are set to the *m* most distinctive local features (here: bright 'blobs') found in the image. Real-number registers are set to the center coordinates of corresponding masks.
2. *Execution*: the operations encoded by *I* are carried out one by one, with intermediate results stored in registers.
3. *Interpretation*: the scalar values $y_j(I,X)$, $j=1,\dots,m$, contained in the *m* real-value registers are interpreted as the output yielded by *I* for image *X*. The values are gathered to form an individual's output vector

$$\mathbf{y}(I,X) = \langle y_1(I,X),\dots,y_m(I,X)\rangle, \tag{7.5}$$

that is subject to further processing described in Section 7.3.

7.2.3 Architecture of the Recognition System

The overall recognition system consists of: (i) the best feature extraction procedures O^* constructed using the approach described in Sections 7.3 and 7.4, and (ii) classifiers trained using those features.

We incorporate a multi-agent methodology that aims to compensate for the suboptimal character of representations elaborated by the evolutionary process and allows us to boost the overall performance.

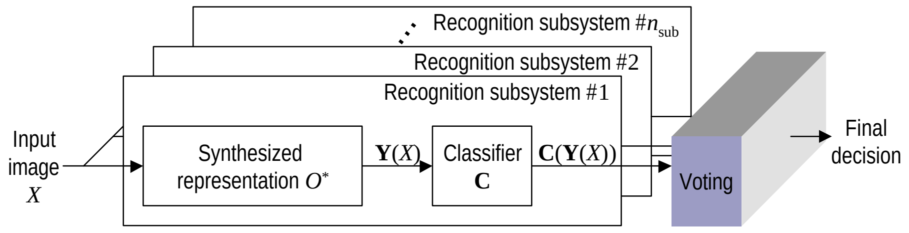


Figure 7.3. The top-level architecture of recognition system.

The basic prerequisite for the agents' fusion to become beneficial is their *diversification*. This may be ensured by using homogenous agents with different parameter settings, homogenous agents with different training data (e.g., bagging [65]), heterogeneous agents, etc. Here, the diversification is naturally provided by the random nature of the genetic search. In particular, we run *many* genetic searches that start from different initial states (initial populations). The best representation O^* evolved in each run becomes a part of a single *subsystem* in the recognition system's architecture (see Figure 7.3). Each subsystem has two major components: (i) a representation O^*, and (ii) a classifier **C** trained using that representation. As this classifier training is done once per subsystem, a more sophisticated classifier **C** may be used here (as compared to the classifier C_{fit} used in the evaluation function).

The subsystems process the input image X independently and output recognition decisions that are further aggregated by a simple majority voting procedure into the final decision. The subsystems are therefore homogenous as far as the structure is concerned; they only differ in the features extracted from the input image and the decisions made. The number of subsystems n_{sub} is a parameter set by the designer.

7.3 Experimental Results

The primary objective of the computational experiment is to test the scalability of the approach with respect to the number of decision classes and its sensitivity to various types of object distortions. As an experimental testbed, we choose the demanding task of object recognition in synthetic aperture radar (SAR) images.

There are several difficulties that make recognition in this modality extremely hard:

- poor visibility of objects – usually only prominent scattering centers are visible,
- low persistence of features under rotation, and
- high levels of noise.

The data source is the MSTAR public database [58] containing real images of several objects taken at different azimuths and at 1-foot spatial resolution. From the original complex (2-channel) SAR images, we extract the magnitude component and crop it to 48×48 pixels. No other form of preprocessing is applied.

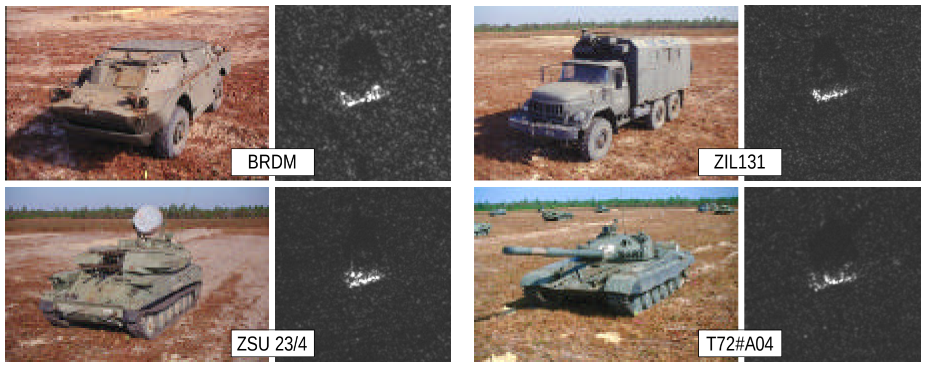


Figure 7.4. Selected objects and their SAR images used in the learning experiment.

The following parameter settings are used for each coevolutionary run: number of subsystems n_{sub}: 10; classifier C_{fit} used for feature set evaluation: decision tree inducer C4.5 [63]; mutation operator: one-point, probability 0.1; crossover operator: one-point, probability 1.0, cutting allowed at every point; selection operator: tournament selection with tournament pool size = 5; number of registers (image and numeric) m: 2; number of populations n: 4; genome length: 40 bytes (10 operations); single population size: 200 individuals; time limit for evolutionary search: 4000 seconds (Pentium PC 1.4 GHz processor).

A *compound* classifier **C** is used to boost the recognition performance. In particular, **C** implements the '1-vs.-all' scheme, i.e. it is composed of l *base classifiers* (where l is the number of decision classes), each of them working as a binary (two-class) discriminator between a single decision class and all the remaining classes. To aggregate their outputs, a simple decision rule is used that yields final class assignment only if the base classifiers are consistent and indicate a single decision class. With this strict rule, any inconsistency among the base classifiers (i.e., no class indicated or more than one class indicated) disables univocal decision and the example remains unclassified (assigned to 'No decision' category).

The system's performance is measured using different base classifiers (if not stated otherwise, the classifier uses default parameter settings as specified in [62]):

- support vector machine with polynomial kernels of degree 3 (trained using sequential minimal optimization algorithm [68] with complexity parameter set to 10),
- nonlinear neural networks with sigmoidal units trained using backpropagation algorithm with momentum,
- C4.5 decision tree inducer [69].

7.3.1 Scalability

To investigate the scalability of the proposed approach w.r.t. to the problem size, we use several datasets with increasing numbers of decision classes for a 15-deg. depression angle, starting from l=2 decision classes: BRDM2 and ZSU. Consecutive problems are created by adding the decision classes up to l=8 in the following order: T62, Zil131, a variant A04 of T72 (T72#A04 in short), 2S1, BMP2#9563, and BTR70#C71.

For i^{th} decision class, its representation D_i in the training data D consists of *two* subsets of images sampled uniformly from the original MSTAR database with respect to a 6-degree azimuth step. Training set D, therefore, always contains 2*(360/6)=120 images from each decision class, so its total size is 120*l. The corresponding test set T contains all the remaining images (for a given object and elevation angle) from the original MSTAR collection. In this way, the training and test sets are strictly disjoint. Moreover, the learning task is well represented by the training set as far as the azimuth is concerned. Therefore, there is no need for multiple train-and-test procedures here and the results presented in the following all use this single particular partitioning of MSTAR data.

Let n_c, n_e, and n_u, denote respectively the numbers of test objects correctly classified, erroneously classified, and unclassified by the recognition system. Figure 7.5(a) presents the *true positive rate*, i.e. $P_{tp}=n_c/(n_c+n_e+n_u)$, also known as *probability of correct identification* (PCI), as a function of the number of decision classes. It can be observed, that the scalability depends heavily on the base classifier, and that SVM clearly outperforms its rivals. For this base classifier, as new decision classes are added to the problem, the recognition performance gradually decreases. The major drop-offs occur when T72 tank and 2S1 self-propelled gun (classes 5 and 6, respectively), are added to the training data; this is probably due to the fact that these objects are visually similar to each other (e.g., both have gun turrets) and significantly resemble the T62 tank (class 3). On the contrary, introducing consecutive classes 7 and 8 (BMP2 and BTR60) did not affect the performance much; more than this, an improvement of accuracy is even observable for class 7.

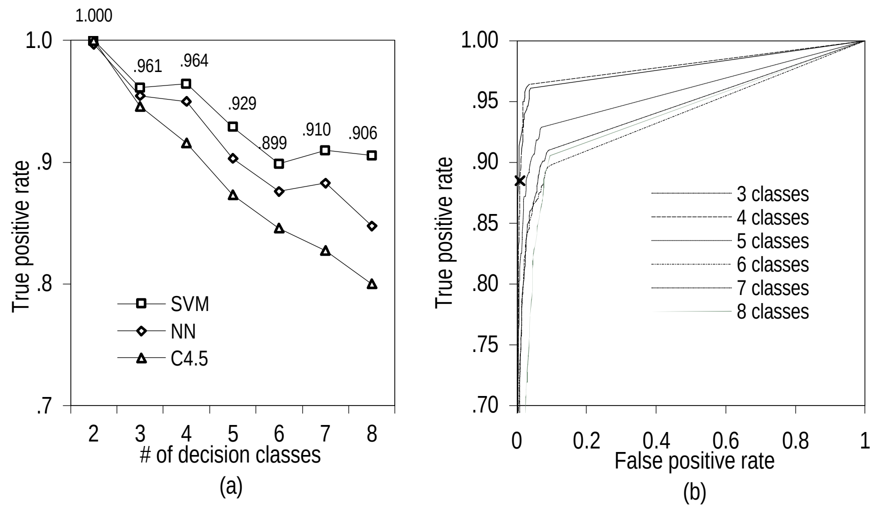


Figure 7.5. (a) Test set recognition ratio as a function of number of decision classes. (b) ROC curves for different number of decision classes (base classifier: SVM).

Figure 7.5(b) shows the *receiver operating characteristics* (ROC) curves obtained, for the recognition systems using SVM as a base classifier, by modifying the confidence threshold that controls whether the classifier votes. The false positive rate is defined here as $P_{fp}=n_e/(n_c+n_e+n_u)$. Again, the results support our method: the curves do not drop rapidly as the false positive rate decreases. Therefore, very high *accuracy of classification*, i.e., $n_c/(n_c+n_e)$, may be obtained when accepting a reasonable *rejection rate* $n_u/(n_c+n_e+n_u)$. For instance, for 4 decision classes, when P_{fp}=0.008, P_{tp}=0.885 (see marked point in Figure 7.5(b)), and, therefore, rejection rate is $1-(P_{fp}+P_{tp})=0.107$, the accuracy of classification equals 0.991.

7.3.2 Object variants

A desirable property of an object recognition system is its ability to recognize different variants of the same object. This task may pose some difficulties, as configurations of vehicles often vary significantly. To provide a comparison with human-designed recognition system, we use the conditions of the experiment reported in [64]. In particular, we synthesized recognition systems using:

- 2 objects: MP2#C21, T72#132,
- 4 objects: MP2#C21, T72#132, BTR70#C71, and ZSU23/4.

For both of these cases, the testing set includes two *other* variants of BMP2 (#9563 and #9566), and two *other* variants of T72 (#812 and #s7).

The results of the test set evaluation shown in the confusion matrices (Table 7.1) suggest that, even when the recognized objects differ significantly from the models provided in the training data, the approach is still able to maintain high performance. Here the true positive rate P_{tp} equals 0.804 and 0.793, for 2- and 4-class systems, respectively. For the cases where a decision can be made (83.3% and 89.2%, respectively), the values of classification accuracy, 0.966 and 0.940, respectively, are comparable to the forced recognition results of the human-designed recognition algorithms reported in [64], which are 0.958 and 0.942, respectively. Note that in the test, we have not used 'confusers', i.e. test images from different classes that those present in the training set, as opposed to [64], where BRDM2 armored personnel carrier has been used for that purpose.

Table 7.1. Confusion matrices for recognition of object variants.

		Predicted class							
		2-class system			*4-class system*				
Test objects		BMP2	T72	No	BMP2	T72	BTR	ZSU	No
Object	*Serial #*	[#C21]	[#132]	decision	[#C21]	[#132]	[#C71]	[#d08]	decision
BMP2	[#9563,9566]	**295**	18	78	**293**	27	27	1	43
T72	[#812,s7]	4	**330**	52	12	**323**	1	9	41

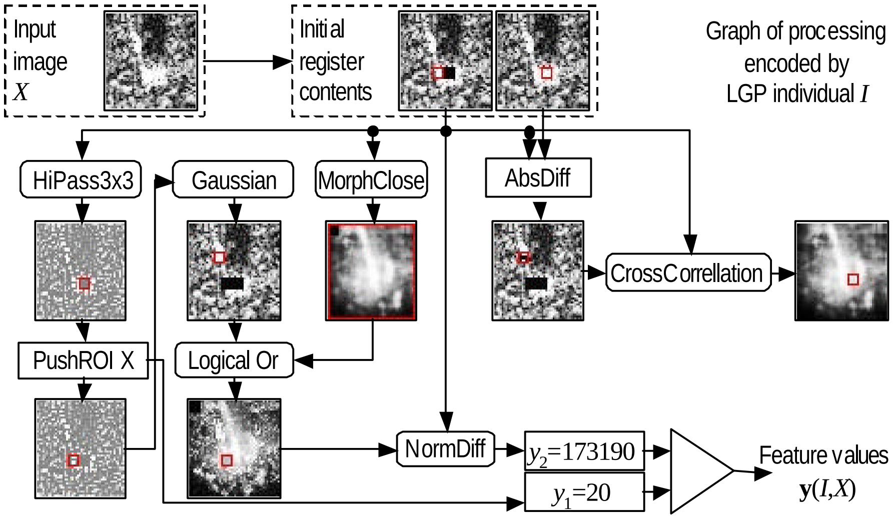


Figure 7.6. Processing carried out by one of the evolved procedures shown as a graph (small rectangles in images depict masks; boxes: local operations; rounded boxes: global operations).

7.4 Summary

In this contribution, we provide experimental evidence for the possibility of synthesizing, without or with little human intervention, a feature-based recognition system, which recognizes 3D objects at the performance level that can be comparable to handcrafted solutions. Let us emphasize that these encouraging results are obtained in the demanding field of SAR imagery, where the acquired images only roughly depict the underlying 3D structure of the object.

There are several major factors that contribute to the overall high performance of the approach. First of all, the paradigm of *coevolution* allows us to decompose the task of representation (feature set) construction into several semi-independent, cooperating subtasks. In this way, we exploit the inherent modularity of the learning process, without the need of specifying explicit objectives for each developed feature extraction procedure. Secondly, the approach manipulates LGP-encoded feature extraction *procedures*, as opposed to most approaches, which are usually limited to learning meant as parameter optimization. This allows for learning sophisticated features, which are novel and sometimes very different from expert's intuition, as may be seen from example shown in Figure 7.6. And thirdly, the *fusion at feature and decision level* helps us to aggregate sometimes contradictory information sources and build a recognition system that is comparable to human-designed system performance with a bunch of simple components at hand.

Chapter 8

Conclusions

This report investigates the efficacy of evolutionary computation such as genetic programming, genetic algorithms and linear genetic programming in learning and selecting features for object detection and object recognition. The reason for incorporating learning into object detection and recognition is to avoid the time consuming process of feature generation and selection. With learning incorporated, an object detection and recognition system

(a) can automatically explore many unconventional features that may yield exceptionally good detection and recognition performances in some cases, thus, overcoming human expert limitation of concentrating only on a small number of conventional features; and

(b) becomes more flexible and is able to automatically generate features on the fly particularly effective to the type of objects and images encountered.

The ultimate goal is to lower the cost of designing object detection and recognition systems and build more robust and flexible systems with human-competitive performance.

The key contributions of this research are:

- We investigate the effectiveness of genetic programming in synthesizing composite operators and composite features for object detection. We show that GP is effective in synthesizing composite operators based on domain-independent primitive operators and domain-independent primitive feature images that can be easily generated from the original image for object detection. The synthesized composite operators can be applied to other testing images similar to the training image. The composite features learned by GP are much more effective than the human-designed primitive features from which they are built. The GP learned composite features may not be imagined by human experts, since these unconventional features are very difficult, if not impossible, to be explained by human experts. Thus, the learning method is of great help in the design of object detection and recognition systems.
- We design an MDL-based fitness function and smart GP operators to improve the efficiency of genetic programming. MDL-based fitness function is

proposed to address the well-known code bloat problem of GP. The MDL-based fitness function takes the size of a composite operator into the fitness evaluation process to prevent composite operators from growing too large without setting a hard limit on the size of a composite operator, imposing relatively less restrictions on the GP search and greatly improving the GP efficiency. To further improve the efficiency of genetic programming, smart crossover and smart mutation are proposed to identify and prevent the effective components of composite operators from being disrupted by destructive crossover and mutation. Also, a public library is set up to keep effective components for later reuse. Compared to traditional genetic programming, the smart GP driven by the MDL-based fitness function and equipped with smart crossover and smart mutation synthesizes composite operators with better performance and smaller size, reducing the computational expense during recognition and the possibility of overfitting the training images.

- We propose an MDL-based fitness function to drive GA in the selection of features for object detection and recognition. The performance of the MDL-based fitness function is compared with those of three other fitness functions. The MDL-based fitness function balances the number of feature selected and the recognition error rate very well and it is the best fitness function compared to other three functions used in the literature. With fewer features selected, the computational expenses and the possibility of overfitting the training data are reduced.

- We propose a coevolutionary genetic programming (CGP) approach to learn composite features for object recognition. The knowledge about the problem domain is incorporated in primitive features that are used as terminals in the synthesis of composite features by CGP using domain independent primitive operators. The motivation for using CGP is to overcome the limitations of human experts who consider only a small number of conventional combinations of primitive features during synthesis. CGP, on the other hand, can try a very large number of unconventional combinations and these unconventional combinations yield exceptionally good results in some cases. Our experimental results with real synthetic aperture radar (SAR) images show that CGP can learn good composite features. We show results to distinguish objects from clutter and to distinguish objects that belong to several classes.

- We propose linear genetic programming in conjunction with cooperative coevolution for feature synthesis for object recognition. General-purpose image processing/computer vision libraries are used for feature synthesis. We show that coevolution is more efficient than genetic algorithms. We consider various strategies for multi-class target recognition. We consider cooperation at various levels and show recognition results for eight classes of MSTAR dataset.

In conclusion, evolutionary computation has a great promise and a strong potential to automate the design of target and pattern recognition systems and provide a human competitive performance.

Bibliography

[1] A. Teller, “Algorithm evolution with internal reinforcement for signal understanding,” *PhD thesis*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1998.

[2] A. Ghosh and S. Tsutsui (Eds.), “Advances in Evolutionary Computing – Theory and Application,” Springer-Verlag, 2003.

[3] B. Bhanu, D. Dudgeon, E. Zelnio, A. Rosenfeld and D. Casasent (Eds.), “Special Issue on Automatic Target Recognition,” *IEEE Transactions on Image Processing,* Vol. 6, No. 1, January 1997.

[4] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs,* MIT Press, 1994.

[5] C. Harris and B. Buxton, “Evolving edge detectors with genetic programming,” *Proceedings of the Genetic Programming, 1st Annual Conference*, pp. 309-314, Cambridge, MA, USA, MIT Press, 1996.

[6] R. Poli, “Genetic programming for feature detection and image segmentation,” in *Evolutionary Computation*, T. C. Forgarty (Ed.), pp. 110-125, 1996.

[7] B. Bhanu and Y. Lin, “Learning composite operators for object detection”, *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1003-1010, July 2002.

[8] S. A. Stanhope and J. M. Daida, “Genetic programming for automatic target classification and recognition in synthetic aperture radar imagery,” *Proceeding Conference on Evolutionary Programming VII.*, pp. 735-744, 1998.

[9] D. Howard, S. C. Roberts, and R. Brankin, “Target detection in SAR imagery by genetic programming,” *Advances in Engineering Software*, Vol. 30, No. 5, pp. 303-311, Elsevier, May 1999.

[10] S. C. Roberts and D. Howard, “Evolution of vehicle detectors for infrared line scan imagery,” *Proceedings of the Evolutionary Image Analysis, Signal Processing and Telecommunications, First European Workshops, EvoIASP’99 and EuroEcTel’99*, Berlin, Germany, pp. 110-125, Springer-Verlag, 1999.

[11] B. Bhanu and S. Fonder, “Learning-integrated interactive image segmentation,” chapter in *Advances in Evolutionary Computing – Theory and Application*, A. Ghosh and S. Tsutsui (Eds.), pp. 863 – 895, Springer-Verlag, 2003.

[12] J. Rissanen, “A Universal Prior for Integers and Estimation by Minimum Description Length,” *Ann. of Statist*, Vol. 11, No. 2, pp. 416-431, 1983.

[13] W. Tackett, “Recombination selection, and the genetic construction of computer programs,” *Ph.D thesis*, University of Southern California, Department of Electrical Engineering-Systems, 1994.

[14] P. D’haeseleer, “Context preserving crossover in genetic programming,” *Proceedings of the IEEE World Congress on Computational Intelligence*, Vol. 1, pp. 256 – 261, 1994.

[15] P. Smith, “Conjugation – A bacterially inspired form of genetic recombination,” In J. R. Koza (Ed.), *Late Breaking Papers at the Genetic Programming Conference*, pp. 167 – 176, 1996.

[16] T. Ito, H. Iba and S. Sato, “Depth-dependent crossover for genetic programming,” *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 775-780, 1998.

[17] B. Bhanu and Y. Lin, “Learning feature agents for extracting terrain regions in remotely sensed images,” *Proceedings of the 2nd Pattern Recognition for Remote Sensing Workshop*, pp 1-6, August 12, 2002.

[18] W. B. Langdon and R. Poli, “Foundations of Genetic Programming,” Springer, 2001.

[19] D. Kreithen, S. Halversen and G. Owirka, “Discriminating targets from clutter,” *Lincoln Laboratory Journal*, Vol. 6, No. 1, pp. 25–52, Spring 1993.

[20] S. Cagnoni, A. Dobrzeniecki, R. Poli and J. Yanch, “Genetic algorithm-based interactive segmentation of 3D medical images,” *Image and Vision Computing,* Vol. 17, No. 12, pp. 881-895, October 1999.

[21] B. Bhanu and T. Poggio (Eds.), “Special Section on Machine Learning in Computer Vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 16, No. 9, September 1994.

[22] M. Rizki, L. Tamburino and M. Zmuda, “Multi-resolution feature extraction from Gabor filtered images,” *Proceedings of the IEEE National Aerospace and Electronics Conference*, pp. 24-28, Dayton, OH, USA, May 1993.

[23] A. Katz and P. Thrift, “Generating image filters for target recognition by genetic learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 16, No. 9, September 1994.

[24] E. Ozcan and C. Mohan, “Partial shape matching using genetic algorithms,” *Pattern Recognition Letters*, Vol. 18, pp. 987-992, 1997.

[25] R. Srikanth, R. George, N. Warsi, D. Prabhu, F. Petry and B. Buckles, “A variable-length genetic algorithm for clustering and classification,” *Pattern Recognition Letters*, Vol. 16, pp. 789-800, 1995.

[26] W. Punch and E. Goodman, "Further research on feature selection and classification using genetic algorithms," *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 557-564, 1993.

[27] B. Bhanu and S. Lee, *Genetic Learning for Adaptive Image Segmentation*, Kluwer Academic Publishers, 1994.

[28] C. Emmanouilidis, A. Hunter, J. MacIntyre and C. Cox, "Multiple-criteria genetic algorithms for feature selection in neuro-fuzzy modeling," *Proceedings of the International Joint Conference on Neural Networks,* Vol. 6, pp. 4387-4392, 1999.

[29] P. Estevez and R. Caballero, "A niching genetic algorithm for selecting features for neural classifiers," *Proceedings of the 8th International Conference on Artificial Neural Networks*, Vol. 1, pp. 311-316, Springer-Verlag, 1998.

[30] F. Rhee and Y. Lee, "Unsupervised feature selection using a fuzzy-genetic algorithm," *Proceedings of the IEEE International Fuzzy Systems Conference,* Vol. 3, pp. 1266-1269, 1999.

[31] K. Matsui, Y. Suganami and Y. Kosugi, "Feature selection by genetic algorithm for MRI segmentation," *Systems and Computers in Japan*, Vol. 30, No. 7, pp. 69-78, Scripta technical, June 30, 1999.

[32] J. Quinlan and R. Rivest, "Inferring decision tree using the minimum description length principle," *Information and Computation*, Vol. 80, pp. 227-248, 1989.

[33] Q. Gao, M. Li and P. Vitanyi, "Applying MDL to learn best model granularity," *Artificial Intelligence*, Vol. 121, pp. 1-29, 2000.

[34] L. Novak, G. Owirka and C. Netishen, "Performance of a high-resolution polarimetric SAR automatic target recognition system," *Lincoln Laboratory Journal*, Vol. 6, No. 1, pp. 11–24, Spring 1993.

[35] L. Novak, M. Burl and W. Irving, "Optimal polarimetric processing for enhanced target detection," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 29, pp. 234-244, 1993.

[36] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recognition Letters*, Vol. 10, pp. 335-347, November 1989.

[37] S. Halversen, "Calculating the orientation of a rectangular target in SAR imagery," *Proceedings of the IEEE National Aerospace and Electronics Conference*, pp. 260-264, May 1992.

[38] M. Ebner and A. Zell, "Evolving a task specific image operator," *Proceedings of the Evolutionary Image Analysis, Signal Processing and*

Telecommunications, First European Workshops, EvoIASP'99 and EuroEcTel'99, Berlin, Germany, pp. 74-89, Springer-Verlag, 1999.

[39] Y. Lin and B. Bhanu, "Discovering operators and features for object detection," *Proceedings of the 16th International Conference on Pattern Recognition*, Vol. 3, pp. 339-342, August 2002.

[40] K. Krawiec and B. Bhanu, "Visual learning by evolutionary feature synthesis," *International Conference on Machine Learning*, pp. 376-383, 2003.

[41] K. Krawiec and B. Bhanu, "Coevolution and linear genetic programming for visual learning," *Genetic and Evolutionary Computation Conference*, Part I., pp. 332-343, 2003.

[42] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, 1999.

[43] W. Banzhaf, P. Nordin, R. Keller and F. Francone, *Genetic Programming, an Introduction on the Automatic Evolution of Computer Programs and its Application,* San Francisco, Morgan Kaufmann, 1998.

[44] B. Bhanu and K. Krawiec, "Coevolutionary construction of features for transformation of representation in machine learning," *Proceedings of the Workshop on Coevolution, Genetic and Evolutionary Computation Conference*, pp. 249-254, 2002.

[45] B. Draper, A. Hanson and E. Riseman, "Learning blackboard-based scheduling algorithms for computer vision," *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 7, pp. 309-328, 1993.

[46] Intel® image processing library, *Reference manual,* Intel Corporation, 2000.

[47] Intel® image processing library, *Reference manual,* Intel Corporation, 2001.

[48] M. P. Johnson, "Evolving visual routines," *Master's Thesis*, Massachusetts Institute of Technology, 1995.

[49] K. Krawiec, "Pairwise comparison of hypotheses in evolutionary learning," In: C. E. Brodley and A. Pohoreckyj Danyluk (Eds.), *Proceedings of the Eighteenth International Conference on Machine Learning,* pp. 266-273, 2001.

[50] S. Luke, "ECJ Evolutionary Computation System," http://www.cs.umd.edu/projects/plus/ec/ecj/, 2002.

[51] M. A. Maloof, P. Langley, T. O. Binford, R. Nevatia and S. Sage, "Improved rooftop detection in aerial images with machine learning," *Machine Learning,* Vol. 53, No. 1/2, 2003.

[52] C. J. Matheus, "A constructive induction framework," *Proceedings of the Sixth International Workshop on Machine Learning,* pp. 474-475, 1989.

[53] J. Peng and B. Bhanu, “Closed-loop object recognition using reinforcement learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, pp. 139-154, 1998.

[54] J. Peng and B. Bhanu, “Delayed reinforcement learning for adaptive image segmentation and feature extraction,” *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 28, pp. 482-488, August 1998.

[55] R. Poli, “Genetic programming for image analysis,” *Technical Report CSRP-96-1*, University of Birmingham, 1996.

[56] M. A. Potter and K. A. De Jong, “Cooperative coevolution: an architecture for evolving coadapted subcomponents,” *Evolutionary Computation*, Vol. 8, pp. 1-29, 2000.

[57] M. Rizki, M. Zmuda and L. Tamburino, “Evolving pattern recognition systems,” *Proceedings of the IEEE International Conference on Evolutionary Computation,* Vol. 6, pp. 594-609, 2002.

[58] T. Ross, S. Worell, V. Velten, J. Mossing and M. Bryant, “Standard SAR ATR evaluation experiments using the MSTAR public release data set,” *SPIE Proceedings: Algorithms for Synthetic Aperture Radar Imagery V.*, Vol. 3370, pp. 566-573, Orlando, FL, 1998.

[59] J. Segen, “GEST: A learning computer vision system that recognizes hand gestures,” In R.S. Michalski and G. Tecuci (Eds.), *Machine learning, A Multistrategy Approach,* Vol. 4, pp. 621-634, Morgan Kaufmann, 1994.

[60] A. Teller and M. Veloso, “A controlled experiment: evolution for learning difficult image classification,” *Lecture Notes in Computer Science,* Vol. 990, pp. 165-185, 1995.

[61] R. P. Wiegand, W. C. Liles and K. A. De Jong, “An empirical analysis of collaboration methods in cooperative coevolutionary algorithms,” *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 1235-1242, Morgan Kaufmann, 2001.

[62] I. H. Witten and E. Frank, *Data mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 1999.

[63] D. Wolpert and W. G. Macready, “No free lunch theorems for optimization” *IEEE Transactions on Evolutionary Computation*, Vol. 1, pp. 67-82, 1997.

[64] B. Bhanu and G. Jones, “Increasing the discrimination of SAR recognition models,” *Optical Engineering,* Vol. 12, pp. 3298–3306, 2002.

[65] L. Breiman, “Bagging predictors,” *Machine Learning,* Vol. 24, pp. 123–140, 1996.

[66] B. Draper, A. Hanson and E. Riseman, “Knowledge-directed vision: control, learning and integration, *Proceedings of the IEEE,* Vol. 84, pp. 1625-1637, 1996.

[67] K. Krawiec, “On the use of pair wise comparison of hypotheses in evolutionary learning applied to learning from visual examples,” In P. Perner (Ed.), *Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Artificial Intelligence,* Vol. 2123, pp. 307-321, Springer-Verlag, 2001.

[68] J. Platt, “Fast training of support vector machines using sequential minimal optimization,” In B. Schölkopf, C. Burges and A. Smola, (Eds.), *Advances in Kernel Methods - Support Vector Learning*, MIT Press, 1998.

[69] J. R. Quinlan, *C4.5: Programs for Machine Learning,* Morgan Kaufmann, San Mateo, California, 1992.

[70] Y. Lin and B. Bhanu, “Learning composite features for object recognition,” Genetic and Evolutionary Computation Conference, Part II, pp. 2227-2239, 2003.